A Decidable Class of Nested Iterated Schemata (extended version)

Vincent Aravantinos, Ricardo Caferra, and Nicolas Peltier

Grenoble University (LIG/CNRS)

Abstract. Many problems can be specified by patterns of propositional formulae depending on a parameter, e.g. the specification of a circuit usually depends on the number of bits of its input. We define a logic whose formulae, called *iterated schemata*, allow to express such patterns. Schemata extend propositional logic with indexed propositions, e.g. P_{i}, $P_{\mathsf{i}+1}$, P_1 or P_{n}, and with generalized connectives, e.g. $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}}$ or $\bigvee_{\mathsf{i}=1}^{\mathsf{n}}$ (called *iterations*) where n is an (unbound) integer variable called a *parameter*. The expressive power of iterated schemata is strictly greater than propositional logic: it is even out of the scope of first-order logic. We define a proof procedure, called DPLL*, that can prove that a schema is satisfiable for at least one value of its parameter, in the spirit of the DPLL procedure [12]. However the converse problem, i.e. proving that a schema is unsatisfiable *for every value of the parameter*, is undecidable [2] so DPLL* does not terminate in general. Still, we prove that DPLL* terminates for schemata of a syntactic subclass called *regularly nested*. This is the first non trivial class for which DPLL* is proved to terminate. Furthermore the class of regularly nested schemata is the first decidable class to allow nesting of iterations, i.e. to allow schemata of the form $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}}(\bigwedge_{\mathsf{j}=1}^{\mathsf{n}}\dots)$.

1 Introduction

The specification of problems in propositional logic often leads to propositional formulae that depend on a parameter: the n-queens problem depends on n, the pigeonhole problem depends on the number of considered pigeons, a circuit may depend on the number of bits of its input, etc. Consider for instance a specification of a carry propagate adder circuit i.e. a circuit that takes as input two n-bit vectors and computes their sum:

$$Adder \stackrel{\text{def}}{=} \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} Sum_{\mathsf{i}} \wedge \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} Carry_{\mathsf{i}} \wedge \neg C_1$$

where:

$$Sum_{\mathsf{i}} \stackrel{\text{def}}{=} S_{\mathsf{i}} \Leftrightarrow (A_{\mathsf{i}} \oplus B_{\mathsf{i}}) \oplus C_{\mathsf{i}}$$

$$Carry_{\mathsf{i}} \stackrel{\text{def}}{=} C_{\mathsf{i}+1} \Leftrightarrow (A_{\mathsf{i}} \wedge B_{\mathsf{i}}) \vee (B_{\mathsf{i}} \wedge C_{\mathsf{i}}) \vee (A_{\mathsf{i}} \wedge C_{\mathsf{i}})$$

$\oplus$ denotes the exclusive OR

$A_1, \ldots, A_{\mathsf{n}}$ denotes the first operand of the circuit

$B_1, \ldots, B_{\mathsf{n}}$ denotes the second operand of the circuit

$S_1, \ldots, S_{\mathsf{n}}$ denotes the output (the **S**um) of the circuit

$C_1, \ldots, C_{\mathsf{n}}$ denotes the intermediate **C**arries of the circuit

Presently, automated reasoning on such specifications requires that we give a concrete value to the parameter n. Besides the obvious loss of generality, this instantiation hides the *structure* of the initial problem which can be however a useful information when reasoning about such specifications: the structure of the proof can in many cases be *guided* by the structure of the original specification. This gave us the idea to consider parameterized formulae at the object level and to design a logic to reason about them.

Notice that schemata not only arise naturally from practical problems, but also have a deep conceptual interpretation, putting bridges between logic and computation. As well as first or higher-order logic abstracts from propositional logic via *quantification*, schemata allow to abstract via *computation*. Indeed, a schema can be considered as a very specific algorithm taking as input a value for the parameter and generating a propositional formula depending on this value. So a schema can be seen as an algorithm whose codomain is the set of propositional formulae (its domain is the set of integers in this presentation, but one can imagine any type of parameter). Thus schemata can be seen as a different – and complementary – way to abstract from propositional logic.

If we want to prove, e.g. that the implementation of a parameterized specification is correct, we need to prove that the corresponding schema is valid *for every value of the parameter*. As usual we actually deal with unsatisfiability: we say that a schema is *unsatisfiable* iff every propositional formula obtained by giving a value to the parameter is unsatisfiable. In [2] we introduced a first proof procedure for propositional schemata, called STAB. Notice that there is an easy way to systematically look for a counter-example (i.e. find a value of the parameter for which the schema is satisfiable): we can just enumerate all the values and check the satisfiability of the corresponding formula with a SAT solver. However this naive procedure does not terminate when the schema is unsatisfiable. On the other hand, STAB not only terminates (and much more efficiently) when the schema is satisfiable, but it can also terminate when the schema is unsatisfiable. However it still *does not terminate in general*, as we proved that the (un)satisfiability problem is undecidable for schemata [2]. As a consequence there cannot exist a complete calculus for schemata (the set of unsatisfiable schemata is not recursively enumerable). Still, we proved that STAB terminates for a particular class of schemata, called *regular*, which is thus decidable (this class contains the carry propagate adder described previously).

An important restriction of the class of regular schemata is that it cannot contain nested iterations, e.g. $\bigvee_{i=1}^{n} \bigvee_{j=1}^{n} P_i \Rightarrow Q_j$. Nested iterations occur frequently in the specification of practical problems. We take the example of a binary multiplier which computes the product of two bit vectors $A = (A_1, \ldots, A_n)$ and $B = (B_1, \ldots, B_n)$ using the following decomposition:

$$A.B = A.\sum_{i=1}^{n} B_i.2^{i-1} = \sum_{i=1}^{n} A.B_i.2^{i-1}$$

The circuit is mainly an iterated sum:

$$\text{“}S^1 = 0\text{”} \wedge \bigwedge_{i=1}^{n} (B_i \Rightarrow Add(S^i, A.2^{i-1}, S^{i+1})) \wedge (\neg B_i \Rightarrow (S^{i+1} \Leftrightarrow S^i))$$

where S^i denotes the i^{th} partial sum (hence S^n denotes the final result) and $Add(x, y, z)$ denotes any schema specifying a circuit which computes the sum z of x and y (for instance the previous *Adder* schema). We express “$S^1 = 0$” by $\bigwedge_{i=1}^{n} \neg S_i^1$, and “$A.2^{i-1}$” by the bit vector $Sh^i = (Sh_1^i, \ldots, Sh_{2n}^i)$ (Sh for *Sh*ift):

$$\left(\bigwedge_{j=1}^{n} Sh_j^1 \Leftrightarrow A_j\right) \wedge \left(\bigwedge_{j=n}^{2n} \neg Sh_j^1\right) \wedge \left(\bigwedge_{i=1}^{n} \neg Sh_1^i \wedge \bigwedge_{j=1}^{2n} (Sh_{j+1}^i \Leftrightarrow Sh_j^i)\right)$$

This schema obviously contains nested iterations[1].

STAB does not terminate in general on such specifications. We introduce in this paper a new proof procedure, called DPLL*, which is an extension of the DPLL procedure [12]. Extending DPLL to schemata is a complex task, because the formulae depend on an *unbounded* number of propositional variables (e.g. $\bigvee_{i=1}^{n} P_i$ “contains” $P_1, \ldots, P_n$). Furthermore, propagating the value given to an atom is not straightforward as in DPLL (in $\bigvee_{i=1}^{n} P_i$ if the value of e.g. P_2 is fixed then we must propagate the assignment to P_i but only in the case where $i = 2$). The main advantage of DPLL* over STAB is that it can operate on subformulae occurring at a deep position in the schema (in contrast to STAB, which only handles root formulae, by applying decomposition rules). This feature turns out to be essential for handling nested iterations. We prove that DPLL* is sound, complete for satisfiability detection and terminates on a class of schemata, called *regularly nested*, which is obtained from regular schemata by removing the restriction on nested iterations.

The paper is organized as follows. Section 2 defines the syntax and semantics of iterated schemata. Section 3 presents the DPLL* proof procedure. Section 4 deals with the detection of cycles in proofs, which is the main tool allowing termination. Section 5 presents the class of *regularly nested schemata*, for which we show that DPLL* terminates. Termination is also proven for some simple derivatives of this class. Section 6 concludes the paper and briefly presents related works.

[1] However it does not belong to the decidable class presented in Section 5.

2 Schemata of Propositional Formulae

Consider the usual signature $\Sigma \stackrel{\text{def}}{=} \{0, s, +, -\}$ and a countable set of *integer variables* denoted by $\mathcal{IV}$. Terms on Σ and $\mathcal{IV}$ are called *linear expressions*, whose set is written $\mathcal{LE}$. As usual we simply write n for $s^n(0)$ $(n > 0)$ and $n.e$ for $e + \cdots + e$ (n times). Linear expressions are considered modulo the usual properties of the arithmetic symbols (e.g. $s(0) + s(s(0)) - 0$ is assumed to be the same as $s(s(s(0)))$ and written 3). Consider the structure $\mathcal{L} \stackrel{\text{def}}{=} \langle \Sigma; =, <, > \rangle$ of linear arithmetic (i.e. same as Presburger arithmetic except that negative integers are also considered). The set of first-order formulae of $\mathcal{L}$ is called the set of *linear constraints* (or in short *constraints*), written $\mathcal{LC}$. As usual, if $C_1, C_2 \in \mathcal{LC}$, we write $C_1 \models C_2$ iff C_2 is a logical consequence of C_1. This relation is well known to be decidable using decision procedures for arithmetic without multiplication see e.g. [10]. It is also well known that linear arithmetic admits quantifier elimination. From now on, closed terms of Σ (i.e. integers) are denoted by n, m, i, j, k, l, linear expressions by e, f, constraints by $C, C_1, C_2, \ldots$ and integer variables by $\mathsf{n}, \mathsf{i}, \mathsf{j}$ (we use this particular typesetting to clearly make the distinction with variables of the meta-language).

To make technical details simpler, and w.l.o.g., only schemata in negative normal form (n.n.f.) are considered. We say that a linear constraint *encloses* a variable i iff there exist $e_1, e_2 \in \mathcal{LE}$ s.t. i does not occur in e_1, e_2 and $C \models e_1 \leq \mathsf{i} \wedge \mathsf{i} \leq e_2$.

Definition 1 (Schemata). *For every $k \in \mathbb{N}$, let $\mathcal{P}_k$ be a set of symbols. The set $\mathfrak{P}$ of* formula patterns *(or, for short,* patterns*) is the smallest set s.t.*

- $\top, \bot \in \mathfrak{P}$
- *If $k \in \mathbb{N}$, $P \in \mathcal{P}_k$ and $e_1, \ldots, e_k \in \mathcal{LE}$ then $P_{e_1,\ldots,e_k} \in \mathfrak{P}$ and $\neg P_{e_1,\ldots,e_k} \in \mathfrak{P}$.*
- *If $\pi_1, \pi_2 \in \mathfrak{P}$ then $\pi_1 \vee \pi_2 \in \mathfrak{P}$ and $\pi_1 \wedge \pi_2 \in \mathfrak{P}$.*
- *If $\pi \in \mathfrak{P}$, $\mathsf{i} \in \mathcal{IV}$, $C \in \mathcal{LC}$ and C encloses i then $\bigwedge_{\mathsf{i}|C} \pi \in \mathfrak{P}$ and $\bigvee_{\mathsf{i}|C} \pi \in \mathfrak{P}$.*

A schema *S is a pair (written as a conjunction) $\pi \wedge C$, where π is a pattern and C is a constraint. C is called the* constraint *of S, written C_S. π is called its* pattern*, written Π_S.*

The first three items define a language that differs from propositional logic only in its atoms which we call *indexed propositions* ($e_1, \ldots, e_k$ are called *indices*). The real novel part is the last item. Patterns of the form $\bigwedge_{\mathsf{i}|C} \pi$ or $\bigvee_{\mathsf{i}|C} \pi$ are called *iterations*. C is called the *domain* of the iteration. In [2] only domains of the form $e_1 \leq \mathsf{i} \wedge \mathsf{i} \leq e_2$ were handled, but as we shall see in Section 3, more general classes of constraints are required to define the DPLL* procedure. If C is unsatisfiable then the iteration is *empty*. Any occurrence of i in π is *bound* by the iteration. A variable occurrence which is not bound is *free*. A variable which has free occurrences in a pattern is a *parameter* of the pattern. A pattern which is just an indexed proposition $P_{e_1,\ldots,e_k}$ is called an *atom*. An atom or the negation of an atom is called a *literal*.

In [2] and [1] a schema was just a pattern, however constraints appear so often that it is more convenient to integrate them to the definition of schema. Informally, a pattern gives a "skeleton" with "holes" and the constraint specifies how the holes can be filled (this choice fits the abstract definition of schema in [11]). This new definition can be emulated with the definition of [1] (as one can see from the upcoming semantics $\bigvee_C \top$ is equivalent to C). In the following we assume w.l.o.g. that C_S entails $\mathsf{n}_1 \geq 0 \wedge \cdots \wedge \mathsf{n}_k \geq 0$ where $\mathsf{n}_1, \ldots, \mathsf{n}_k$ are the parameters of Π_S.

Example 1. S is a schema:

$$S \stackrel{\text{def}}{=} P_1 \wedge \bigwedge_{1 \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n}} (Q_\mathsf{i} \wedge \bigvee_{1 \leq \mathsf{j} \leq \mathsf{n}+1 \wedge \mathsf{i} \neq \mathsf{j}} \neg P_\mathsf{n} \vee P_{\mathsf{j}+1}) \wedge \mathsf{n} \geq 1$$

P_1, Q_i, P_n and $P_{\mathsf{j}+1}$ are indexed propositions. The only iterations of S are:

$$\bigvee_{1 \leq \mathsf{j} \leq \mathsf{n}+1 \wedge \mathsf{i} \neq \mathsf{j}} \neg P_\mathsf{i} \vee P_{\mathsf{i}+1}$$

and

$$\bigwedge_{1 \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n}} (Q_\mathsf{j} \wedge \bigvee_{1 \leq \mathsf{j} \leq \mathsf{n}+1 \wedge \mathsf{i} \neq \mathsf{j}} \neg P_\mathsf{i} \vee P_{\mathsf{i}+1})$$

Their respective domains are $1 \leq \mathsf{j} \leq \mathsf{n}+1 \wedge \mathsf{i} \neq \mathsf{j}$ and $1 \leq \mathsf{i} \leq \mathsf{n}$. n is the only parameter of S. Finally:

$$\Pi_S = P_1 \wedge \bigwedge_{1 \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n}} (Q_\mathsf{i} \wedge \bigvee_{1 \leq \mathsf{j} \leq \mathsf{n}+1 \wedge \mathsf{i} \neq \mathsf{j}} \neg P_\mathsf{n} \vee P_{\mathsf{j}+1})$$

and $C_S = \mathsf{n} \geq 1$.

Schemata are denoted by S, S_1, $S_2 \ldots$, parameters by $\mathsf{n}, \mathsf{n}_1, \mathsf{n}_2 \ldots$, bound variables by i, j. $\Delta_{\mathsf{i}|C}\, S$ and $\nabla_{\mathsf{i}|C}\, S$ denote generic iterations (i.e. $\bigvee_{\mathsf{i}|C} S$ or $\bigwedge_{\mathsf{i}|C} S$), $\triangle$ and $\triangledown$ denote generic binary connectives (i.e. $\vee$ or $\wedge$), finally $\Delta_{\mathsf{i}=e_1}^{e_2}\, S$ denotes $\Delta_{\mathsf{i}|e_1 \leq \mathsf{i} \wedge \mathsf{i} \leq e_2}\, S$.

Let S be a schema and $\Delta_{\mathsf{i}_1|C_1}\, S_1$, …, $\Delta_{\mathsf{i}_k|C_k}\, S_k$ be all the iterations occurring in S. Then $C_S \wedge C_1 \wedge \cdots \wedge C_k$ is called the *constraint context* of S, written Context(S). Notice that Context(S) loses the information on the *binding positions* of variables. This can be annoying if a variable name is bound by two different iterations or if it is both bound and free in the schema. So we assume that all schemata are such that this situation does not hold[2].

Substitutions on integer variables map integer variables to linear arithmetic expressions. We write $[e_1/\mathsf{i}_1, \ldots, e_k/\mathsf{i}_k]$ for the substitution mapping $\mathsf{i}_1, \ldots, \mathsf{i}_k$ to $e_1, \ldots, e_k$ respectively. The application of a substitution σ to an arithmetic expression e, written $e\sigma$, is defined as usual. Substitution application is naturally

[2] The proof system defined in Section 3 preserves this property, except for the rule *Emptiness* which duplicates an iteration; but we may safely assume that the variables of one of the duplicated iterations are renamed so that the desired property is fulfilled.

extended to schemata (notice that bound variables are not replaced). A substitution is *ground* iff it maps integer variables to integers (i.e. ground arithmetic expressions). An *environment* ρ of a schema S is a ground substitution mapping all parameters of S and such that $C_S\rho$ is true.

Definition 2 (Propositional Realization). *Let π be a pattern and ρ a ground substitution. The propositional formula $|\pi|_\rho$ is defined as follows:*

- $|P_{e_1,\ldots,e_k}|_\rho \stackrel{def}{=} P_{e_1\rho,\ldots,e_k\rho}$, $|\neg P_{e_1,\ldots,e_k}|_\rho \stackrel{def}{=} \neg P_{e_1\rho,\ldots,e_k\rho}$,
- $|\top|_\rho \stackrel{def}{=} \top$, $|\bot|_\rho \stackrel{def}{=} \bot$, $|\pi_1 \wedge \pi_2|_\rho \stackrel{def}{=} |\pi_1|_\rho \wedge |\pi_2|_\rho$, $|\pi_1 \vee \pi_2|_\rho \stackrel{def}{=} |\pi_1|_\rho \vee |\pi_2|_\rho$
- $|\bigvee_{\mathsf{i}|C} \pi|_\rho \stackrel{def}{=} \bigvee_{i\in\mathbb{Z}\ s.t.\ C[i/\mathsf{i}]\rho\ is\ valid} |\pi[i/\mathsf{i}]|_{\rho\cup[i/\mathsf{i}]}$
- $|\bigwedge_{\mathsf{i}|C} \pi|_\rho \stackrel{def}{=} \bigwedge_{i\in\mathbb{Z}\ s.t.\ C[i/\mathsf{i}]\rho\ is\ valid} |\pi[i/\mathsf{i}]|_{\rho\cup[i/\mathsf{i}]}$

When ρ is an environment of a schema S, we define $|S|_\rho$ as $|\Pi_S|_\rho$. $|S|_\rho$ is called a propositional realization *of S.*

Notice that $\top, \bot, \vee, \wedge, \neg$ on the right-hand members of equations have their standard *propositional* meanings. $\bigvee$ and $\bigwedge$ on the right-hand members are meta-operators denoting respectively the *propositional* formulae $\cdots \vee \cdots \vee \cdots$ and $\cdots \wedge \cdots \wedge \cdots$ or $\bot$ and $\top$ when the conditions are not verified. On the contrary all those symbols on the left-hand members are *pattern* connectives.

We now make precise the semantics outlined in the introduction. Propositional logic semantics are defined as usual. A *propositional interpretation* of a (propositional) formula ϕ is a function mapping every propositional variable of ϕ to a truth value $true$ or $false$.

Definition 3 (Semantics). *Let S be a schema. An* interpretation $\mathcal{I}$ of the schemata language *is the pair of an environment $\rho_\mathcal{I}$ of S and a propositional interpretation $\mathcal{I}_p$ of $|S|_{\rho_\mathcal{I}}$. A schema S is* true *in $\mathcal{I}$ iff $|S|_{\rho_\mathcal{I}}$ is true in $\mathcal{I}_p$, in which case $\mathcal{I}$ is a* model *of S. S is* satisfiable *iff it has a model.*

Notice that an empty iteration $\bigvee_{\mathsf{i}|C} \pi$ (resp. $\bigwedge_{\mathsf{i}|C} \pi$) is always false (resp. true).

Example 2. Consider the following schema:

$$S \stackrel{\text{def}}{=} P_1 \wedge \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} (P_\mathsf{i} \Rightarrow P_{\mathsf{i}+1}) \wedge \neg P_{\mathsf{n}+1} \wedge \mathsf{n} \geq 0$$

(as usual, $S_1 \Rightarrow S_2$ is a shorthand for $\neg S_1 \vee S_2$). Then

$$\begin{aligned}
|S|_{\mathsf{n}\mapsto 0} &= P_1 \wedge \neg P_1 \\
|S|_{\mathsf{n}\mapsto 1} &= P_1 \wedge (P_1 \Rightarrow P_2) \wedge \neg P_2 \\
|S|_{\mathsf{n}\mapsto 2} &= P_1 \wedge (P_1 \Rightarrow P_2) \wedge (P_2 \Rightarrow P_3) \wedge \neg P_3 \\
&\text{etc.}
\end{aligned}$$

S is clearly unsatisfiable. Notice that $\mathsf{n} \mapsto -k$ is *not an environment* of S for any $k > 0$.

The set of satisfiable schemata is recursively enumerable but not recursive [2]. Hence there cannot be a refutationally complete proof procedure for schemata. Notice that the semantics are different from the ones in [2] and [1] but easily seen to be equivalent.

The next definitions will be useful in the definition of DPLL*. Let ϕ be a propositional formula and L a (propositional) literal. We say that L occurs *positively* in ϕ, written $L \sqsubset \phi$, iff there is an occurrence of L in ϕ which is not in the scope of a negation. As we consider formulae in n.n.f., a negative literal occurs positively in ϕ iff it simply occurs in ϕ.

Definition 4. *Let S be a schema and L a literal s.t. the parameters of L are parameters of S.*

We write $L \sqsubset_\Box S$ iff for every environment ρ of S, $|L|_\rho \sqsubset |S|_\rho$.

We write $L \sqsubset_\Diamond S$ iff there is an environment ρ of S s.t. $|L|_\rho \sqsubset |S|_\rho$.

Example 3. Consider S as in Example 2. We have $P_1 \sqsubset_\Box S$, $P_{\mathsf{n}+1} \sqsubset_\Box S$, $P_2 \not\sqsubset_\Box S$. However $P_2 \sqsubset_\Diamond S$ and $P_2 \sqsubset_\Box (S \wedge \mathsf{n} \geq 1)$. Finally $P_0 \not\sqsubset_\Diamond S$ and $P_{\mathsf{n}+2} \not\sqsubset_\Diamond S$. Notice that $\neg P_1 \sqsubset_\Diamond S$ as $S_1 \Rightarrow S_2$ is a shorthand for $\neg S_1 \vee S_2$.

Suppose L has the form $P_{e_1,\ldots,e_k}$ (resp. $\neg P_{e_1,\ldots,e_k}$). For a literal $L' \sqsubset S$ of indices $f_1, \ldots, f_k$, $\phi_L(L')$ denotes the formula:

$$\exists \mathsf{i}_1 \ldots \mathsf{i}_n (C_{\mathsf{i}_1} \wedge \cdots \wedge C_{\mathsf{i}_n} \wedge e_1 = f_1 \wedge \cdots \wedge e_k = f_k)$$

where $\mathsf{i}_1, \ldots, \mathsf{i}_n$ are all the bound variables of S occurring in $f_1, \ldots, f_k$ and $C_{\mathsf{i}_1}, \ldots, C_{\mathsf{i}_n}$ are the domains of the iterations binding $\mathsf{i}_1, \ldots, \mathsf{i}_n$. Then $\phi_L(S)$ denotes the following formula:

$$\bigvee \{\phi_L(P_{f_1,\ldots,f_k}) \mid P_{f_1,\ldots,f_k} \sqsubset S\} \quad (\text{resp. } \bigvee \{\phi_L(\neg P_{f_1,\ldots,f_k}) \mid \neg P_{f_1,\ldots,f_k} \sqsubset S\})$$

Proposition 1. *$L \sqsubset_\Box S$ iff $\forall \mathsf{n}_1, \ldots, \mathsf{n}_l (C_S \Rightarrow \phi_L(S))$ is valid, where $\mathsf{n}_1 \ldots \mathsf{n}_l$ are all the parameters of S. $L \sqsubset_\Diamond S$ iff $\exists \mathsf{n}_1, \ldots, \mathsf{n}_l (C_S \wedge \phi_L(S))$ is valid.*

Proof. (Sketch) Let ρ be an environment of S. Assume that L has the form $P_{e_1,\ldots,e_k}$ (the case $\neg P_{e_1,\ldots,e_k}$ is similar). From Definition 2, it is easily seen (by induction on the number of nested iterations) that $|L|_\rho \sqsubset |S|_\rho$ iff there is a literal $P_{f_1,\ldots,f_k} \sqsubset S$ s.t. $L\rho = P_{f_1,\ldots,f_k}(\rho \cup [i_1/\mathsf{i}_1, \ldots, i_n/\mathsf{i}_n])$ where $\mathsf{i}_1, \ldots, \mathsf{i}_n$ are all the bound variables occurring in $f_1, \ldots, f_k$ and $i_1, \ldots, i_n$ are such that s.t. $C_1(\rho \cup [i_1/\mathsf{i}_1, \ldots, i_n/\mathsf{i}_n]), \ldots, C_n(\rho \cup [i_1/\mathsf{i}_1, \ldots, i_n/\mathsf{i}_n])$ are valid. It is then obvious that $|L|_\rho \sqsubset |S|_\rho$ iff $\phi\rho$ is valid. The result follows easily. □

Example 4. Consider S as defined in Example 2. For any expression e, $P_e \sqsubset_\Box S$ (resp. $P_e \sqsubset_\Diamond S$) iff $\forall \mathsf{n}(\mathsf{n} \geq 0) \Rightarrow [e = 1 \vee \exists \mathsf{i}(1 \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} \wedge e = \mathsf{i}) \vee \exists \mathsf{i}(1 \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} \wedge e = \mathsf{i}+1) \vee e = \mathsf{n}+1]$ (resp. $\exists \mathsf{n} \exists \mathsf{i}(\mathsf{n} \geq 0) \wedge (1 \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n}) \wedge (e = 1 \vee e = \mathsf{i} \vee e = \mathsf{i}+1 \vee e = \mathsf{n}+1)$) is valid.

Then, by decidability of linear arithmetic, both $\sqsubset_\Box$ and $\sqsubset_\Diamond$ are decidable. Besides, it is easy to compute the set $\mathcal{L}(S) \stackrel{\text{def}}{=} \{L \mid L \sqsubset_\Box S\}$ for a given schema

S: one just take any propositional realization ϕ of S, and check for every literal $L \sqsubset \phi$ if $L \sqsubset_\square S$. If yes then it belongs to $\mathcal{L}(S)$ otherwise it does not. It is enough to do this with only one propositional realization as for any $L \in \mathcal{L}(S)$, we must have $L \sqsubset \phi$ for *every* propositional realization ϕ.

3 A Proof Procedure: DPLL*

We provide now a set of (sound) deduction rules (in the spirit of the Davis-Putnam-Logemann-Loveland procedure for propositional logic [12]) complete w.r.t. satisfiability (we know that it is not possible to get refutational completeness). Compared to other proof procedures [2] DPLL* allows to rewrite subformulae occurring at deep positions inside a schema — in particular occurring in the scope of iterated connectives: this is crucial to handle nested iterations.

3.1 Extension Rules

DPLL* is a tableaux-like procedure: rules are given to construct a tree whose root is the formula that one wants to refute. The formula is refuted iff all the branches are contradictory.

As usual with tableaux related methods, the aim of branching is to browse the possible interpretations of the schema. As a schema interpretation assigns a truth value to each atom and a number to each parameter, there are two branching rules: one for atoms, called *Propositional splitting* (this rule assigns a value to propositional variables, as the splitting rule in DPLL), and one for parameters, called *Constraint splitting*. However *Constraint splitting* does not give a value to the parameters, but rather restricts their values by refining the constraint of the schema (i.e. C_S), e.g. the parameter can be either greater or lower than a given integer, leading to two branches in the tableaux. Naturally, in order to analyze a schema, one has to investigate the contents of iterations. So a relevant constraint to use for the branching is the one that states the emptiness of some iteration. In the branch where the iteration is empty, we can replace it by its neutral element (i.e. $\top$ for $\bigwedge$ and $\bot$ for $\bigvee$), which is done by *Constraint splitting* (this may also entails the emptiness of some other iterations, and thus their replacement by their neutral elements too, this is handled by *Algebraic simplification*). Then in the branch where the iteration is not empty, we can unfold the iteration: this is done by the *Unfolding* rule.

Iterations might occur in the scope of other iterations. Thus their domains might depend on variables bound by the outer iterations. *Constraint splitting* is of no help in this case, indeed it makes a branching only according to the values of the *parameter*: bound variables are out of its scope. Hence we define the rule *Emptiness* that can make a "deep" branching, i.e. a branching not in the tree, but in the schema itself: it "separates" an iteration into two distinct ones, depending on the constraint stating the emptiness of the inner iteration, e.g. $\bigvee_{\mathsf{i}=1}^{\mathsf{n}} \bigvee_{\mathsf{j}=3}^{\mathsf{i}} P_{\mathsf{i}} \wedge \mathsf{n} \geq 2$ is replaced by $\bigvee_{\mathsf{i}=3}^{\mathsf{n}} \bigvee_{\mathsf{j}=3}^{\mathsf{i}} P_{\mathsf{i}} \vee \bigvee_{\mathsf{i}=1}^{2} \bot \wedge \mathsf{n} \geq 2$. The reader can notice that *Constraint splitting* and *Emptiness* are very similar. It could be

possible to merge both into only one rule e.g. by considering infinite iterations but this would complicate all the formalism for only a little gain in the proof system. Furthermore, *Emptiness* differs "conceptually" from *Constraint splitting* in the sense that its role is not to browse interpretations but only to analyse a formula.

Constraint splitting strongly affects the application of *Propositional splitting*. Indeed *Propositional splitting* only applies on atoms occurring in *every* instance of the schema (which is formalized by Definition 4), and we saw in Example 3 that this depends on the constraint of the schema. Once an atom has been given the value true (resp. false) we can substitute it with $\top$ (resp. $\bot$). However this is not as simple as in the propositional case as this atom may occur in a realization of the schema without occurring in the schema itself (e.g. P_1 in $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}}$ $(\star)$), so we cannot just substitute $\top$ to it. The simplification is performed by the rule *Expansion* which wraps the indexed propositions that are more general than the considered atom (P_{i} in $(\star)$) with an iteration whose domain is a disunification constraint stating that the proposition is distinct from the considered atom (this gives for $(\star)$: $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \bigwedge_{\mathsf{j}|\mathsf{i}\neq 1\wedge \mathsf{j}=0} P_{\mathsf{i}}$). The introduced iteration is very specific because the bound variable always equals 0 (actually this variable is not used and does not even occur in the wrapped proposition but we assign it 0 to satisfy the condition in Definition 1 that it has to be enclosed by the domain). Whereas usual iterations shall be considered as "for loops", this iteration shall be considered as an "if then else". It all makes sense when *Emptiness* or *Constraint splitting* is applied: *if* the condition holds (i.e. if the wrapped indexed proposition differs from the atom) *then* the contents of the iteration hold (i.e. we keep the indexed proposition as is) *else* the iteration is empty (i.e. we replace it by its neutral element). In $(\star)$, *Emptiness* applies:

$$\bigwedge_{\mathsf{i}|1\leq \mathsf{i}\leq \mathsf{n}\wedge \exists \mathsf{j}(\mathsf{i}\neq 1\wedge \mathsf{j}=0)} \bigwedge_{\mathsf{j}|\mathsf{i}\neq 1\wedge \mathsf{j}=0} P_{\mathsf{i}} \wedge \bigwedge_{\mathsf{i}|1\leq \mathsf{i}\leq \mathsf{n}\wedge \forall \mathsf{j}(\mathsf{i}=1\vee \mathsf{j}\neq 0)} \top$$

(of course the domains can be simplified to allow reader-friendly presentation: $\bigwedge_{\mathsf{i}|2\leq \mathsf{i}\leq \mathsf{n}} \bigwedge_{\mathsf{j}|\mathsf{i}\neq 1\wedge \mathsf{j}=0} P_{\mathsf{i}} \wedge \bigwedge_{\mathsf{i}|\bot} \top$). Then *Algebraic simplification* gives:

$$\bigwedge_{\mathsf{i}|1\leq \mathsf{i}\leq \mathsf{n}\wedge \exists \mathsf{j}(\mathsf{i}\neq 1\wedge \mathsf{j}=0)} P_{\mathsf{i}}$$

i.e. $\bigwedge_{\mathsf{i}=2}^{\mathsf{n}} P_{\mathsf{i}}$, as expected. All this process may seem cumbersome, but it is actually a uniform and powerful way of propagating constraints about nested iterations along the schema. The alternative would be to consider different expansion rules depending on the fact that $f_1, \ldots, f_k$ occur in an iteration or not, which would be rather tedious.

Finally we may know that an iteration is empty without knowing which value of the bound variable satisfies the domain constraint e.g. if a constraint, that we know not to be empty, contains $e \leq \mathsf{i} \wedge f \leq \mathsf{i}$ then how can we know which rank of e or f can indeed be reached? In such cases, the *Interval splitting* rule adds some constraints on the involved expressions to ensure this knowledge.

We now define DPLL* formally.

Definition 5 (Tableau). *A* tableau *is a tree $\mathcal{T}$ s.t. each node α in $\mathcal{T}$ is labeled with a pair $(S_{\mathcal{T}}(\alpha), \mathcal{L}_{\mathcal{T}}(\alpha))$ containing a schema and a finite set of literals.*

If α is the root of the tree then $\mathcal{L}_{\mathcal{T}}(\alpha) = \emptyset$ and $S_{\mathcal{T}}(\alpha)$ is called the *root schema*. The transitive closure of the child-parent relation is written $\prec$. For a set of literals $\mathcal{L}$, $\bigwedge_{\mathcal{L}}$ denotes the pattern $\bigwedge_{L \in \mathcal{L}} L$.

As usual a tableau is generated from another tableau by applying extension rules written $\frac{P}{C}$ (resp. $\frac{P}{C_1 | C_2}$) where P is the premise and C (resp. C_1, C_2) the conclusion(s). Let α be a leaf of a tree $\mathcal{T}$, if the label of α matches the premise then we can *extend* the tableau by adding to α a child (resp. two children) labeled with $C\sigma$ (resp. $C_1\sigma$ and $C_2\sigma$), where σ is the matching substitution. A leaf α is *closed* iff $\Pi_{S_{\mathcal{T}}(\alpha)}$ is equal to $\bot$ or $C_{S_{\mathcal{T}}(\alpha)}$ is unsatisfiable.

When used in a premise, $S[\pi]$ means that the schema π occurs in S; then in a conclusion, $S[\pi']$ denotes S in which π has been substituted with π'.

Definition 6 (DPLL* rules). *The* extension rules *are:*

- Propositional splitting.

$$\frac{(S, \mathcal{L})}{(S, \mathcal{L} \cup P_{e_1,\ldots,e_k}) \mid (S, \mathcal{L} \cup \neg P_{e_1,\ldots,e_k})}$$

  if either $P_{e_1,\ldots,e_k} \sqsubset_\Box S$ or $\neg P_{e_1,\ldots,e_k} \sqsubset_\Box S$, and neither $P_{e_1,\ldots,e_k} \sqsubset_\Diamond \bigwedge_{\mathcal{L}} \wedge C_S$ nor $\neg P_{e_1,\ldots,e_k} \sqsubset_\Diamond \bigwedge_{\mathcal{L}} \wedge C_S$.
- Constraint splitting. *For $(\Delta, \varepsilon) \in \{(\bigwedge, \top), (\bigvee, \bot)\}$:*

$$\frac{(S[\Delta_{\mathsf{i}|C}\, \pi], \mathcal{L})}{(S[\Delta_{\mathsf{i}|C}\, \pi] \wedge \exists \mathsf{i} C, \mathcal{L}) \mid (S[\varepsilon] \wedge \forall \mathsf{i} \neg C, \mathcal{L})}$$

  if $C_S \wedge \forall \mathsf{i} \neg C$ is satisfiable and free variables of C other than i *are parameters.*
- *Rewriting:*

$$\frac{(S_1, \mathcal{L})}{(S_2, \mathcal{L})}$$

  where $C_{S_2} = C_{S_1}$ and $\Pi_{S_1} \to \Pi_{S_2}$ by the following rewrite system:
  - Algebraic simplification. *For every pattern π:*

$$\neg\top \to \bot \quad \pi \wedge \top \to \pi \quad \pi \wedge \bot \to \bot \quad \bigwedge_{\mathsf{i}|C} \top \to \top \quad \pi \wedge \pi \to \pi$$

$$\neg\bot \to \top \quad \pi \vee \top \to \top \quad \pi \vee \bot \to \pi \quad \bigvee_{\mathsf{i}|C} \bot \to \bot \quad \pi \vee \pi \to \pi$$

if $\mathrm{Context}(S_1) \wedge \exists \mathsf{i} C$ *is unsatisfiable:* $\bigwedge_{\mathsf{i}|C} \pi \to \top \quad \bigvee_{\mathsf{i}|C} \pi \to \bot$

if $\mathrm{Context}(S_1) \Rightarrow \exists \mathsf{i} C$ *is valid and π does not contain* i*:* $\Delta_{\mathsf{i}|C}\, \pi \to \pi$

- Unfolding. *For* $(\triangle, \Delta) \in \{(\wedge, \bigwedge), (\vee, \bigvee)\}$*:*

$$\underset{\mathsf{i}|C}{\Delta}\, \pi \to \pi[e/\mathsf{i}] \,\triangle\, \underset{\mathsf{i}|C \wedge \mathsf{i} \neq e}{\Delta}\, \pi \quad \textit{if } \mathrm{Context}(S_1) \Rightarrow C[e/\mathsf{i}] \textit{ is valid}$$

  e can be chosen arbitrarily[3]*.*
- Emptiness. *For* $(\triangle, \Delta) \in \{(\wedge; \bigwedge), (\vee; \bigvee)\}$*,* $(\nabla, \varepsilon) \in \{(\bigwedge; \top), (\bigvee; \bot)\}$*:*

$$\underset{\mathsf{i}|C}{\Delta}(\pi[\underset{\mathsf{i}'|C'}{\nabla}\, \pi']) \to \underset{\mathsf{i}|C \wedge \exists \mathsf{i}' C'}{\Delta}(\pi[\underset{\mathsf{i}'|C'}{\nabla}\, \pi']) \,\triangle\, \underset{\mathsf{i}|C \wedge \forall \mathsf{i}' \neg C'}{\Delta}(\pi[\varepsilon])$$

  if $\mathrm{Context}(S_1) \wedge \forall \mathsf{i}' \neg C'$ *is satisfiable and* i *occurs free in* C'*.*
- Expansion.

$$P_{e_1,\ldots,e_k} \to \bigwedge_{\mathsf{i}|(e_1 \neq f_1 \vee \cdots \vee e_k \neq f_k) \wedge \mathsf{i}=0} P_{e_1,\ldots,e_k} \quad \textit{if } P_{f_1,\ldots,f_k} \in \mathcal{L}$$

$$P_{e_1,\ldots,e_k} \to \bigvee_{\mathsf{i}|(e_1 \neq f_1 \vee \cdots \vee e_k \neq f_k) \wedge \mathsf{i}=0} P_{e_1,\ldots,e_k} \quad \textit{if } \neg P_{f_1,\ldots,f_k} \in \mathcal{L}$$

  if $\mathrm{Context}(S_1) \wedge e_1 = f_1 \wedge \cdots \wedge e_k = f_k$ *is satisfiable.* i *is a fresh variable.*

– Interval splitting. *For* $k, l \in \mathbb{N}$*,* $\Delta \in \{\bigwedge, \bigvee\}$*,* $\lhd \in \{<, \leq, \geq, >\}$*:*

$$\frac{(S[\Delta_{\mathsf{i}|C \wedge k.\mathsf{i} \lhd e_1 \wedge l.\mathsf{i} \lhd e_2}\, \pi], \mathcal{L})}{(S[\Delta_{\mathsf{i}|C \wedge k.\mathsf{i} \lhd e_1}\, \pi] \wedge l.e_1 \lhd k.e_2, \mathcal{L}) \;\Big|\; (S[\Delta_{\mathsf{i}|C \wedge l.\mathsf{i} \lhd e_2}\, \pi] \wedge l.e_1 \not\lhd k.e_2, \mathcal{L})}$$

  if every free variable of C *is either* i *or a parameter, all variables of* e_1, e_2 *are parameters and* $k > 0$*,* $l > 0$*.*

3.2 Looping Detection

The above extension rules do not terminate in general, but this is not surprising as the satisfiability problem is undecidable [2]. Non-termination comes from the fact that iterations can be infinitely unfolded (consider e.g. $\bigvee_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}} \wedge \neg P_{\mathsf{i}}$), thus leading to infinitely many new schemata. However it is often the case that newly obtained schemata have already been seen (up to some relation that remains to be defined) i.e. the procedure is *looping* (e.g. $\bigvee_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}} \wedge \neg P_{\mathsf{i}}$ will generate $\bigvee_{\mathsf{i}=1}^{\mathsf{n}-1} P_{\mathsf{i}} \wedge \neg P_{\mathsf{i}}$, then $\bigvee_{\mathsf{i}=1}^{\mathsf{n}-2} P_{\mathsf{i}} \wedge \neg P_{\mathsf{i}}$, then ... which are all equal up to a shift of n). This is actually an algorithmic interpretation of a proof by mathematical induction. We now define precisely the notion of looping.

We start with a very general definition:

Definition 7 (Looping). *Let* S_1, S_2 *be two schemata having the same parameters* $\mathsf{n}_1, \ldots, \mathsf{n}_k$*, we say that* S_1 loops *on* S_2 *iff for every model* $\mathcal{I}$ *of* S_1 *there is a model* $\mathcal{J}$ *of* S_2 *s.t.* $\rho_{\mathcal{J}}(\mathsf{n}_j) < \rho_{\mathcal{I}}(\mathsf{n}_j)$ *for some* $j \in 1..k$ *and* $\rho_{\mathcal{J}}(\mathsf{n}_l) \leq \rho_{\mathcal{I}}(\mathsf{n}_l)$ *for every* $l \neq j$*. The induced relation among schemata is called the* looping relation*.*

[3] e.g. in Section 5.2 we choose the maximal integer fulfilling the desired property.

Looping is undecidable (e.g. if $S_2 = \bot$ then S_1 loops on S_2 iff S_1 is unsatisfiable). It is trivially transitive. An advantage of Definition 7 is that, contrarily to the definitions found in [2, 1], it is independent of the considered proof procedure. However we still have to make precise the link with DPLL*:

Definition 8. *Let α, β be nodes in a tableau $\mathcal{T}$. $S\mathcal{L}_\mathcal{T}(\alpha)$ denotes the schema $S_\mathcal{T}(\alpha) \wedge \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)}$. Then β* loops *on α iff $S\mathcal{L}_\mathcal{T}(\beta)$ loops on $S\mathcal{L}_\mathcal{T}(\alpha)$.*

Following the terminology of [6], β is called a *bud* node and α its *companion* node. The *Looping* rule closes a leaf that loops on some existing node of the tableau. From now on, DPLL* denotes the extension rules, plus the *Looping* rule.

An example of a tableau generated by DPLL* can be found in Appendix A.

3.3 Soundness and Completeness

Definition 9. *Let $\mathcal{I}$ be an interpretation and α a leaf of a tableau $\mathcal{T}$. We write $\mathcal{I} \models_\mathcal{T} \alpha$ iff $\mathcal{I} \models S\mathcal{L}_\mathcal{T}(\alpha)$ (or simply $\mathcal{I} \models \alpha$ when $\mathcal{T}$ is obvious from the context). We write $\mathcal{I} \models \mathcal{T}$ iff there exists a leaf α in $\mathcal{T}$ s.t. $\mathcal{I} \models \alpha$. The definitions of* model *and* satisfiability *naturally extend.*

Lemma 1. *Let $\mathcal{T}$, $\mathcal{T}'$ be tableaux s.t. $\mathcal{T}'$ is obtained by applying an extension rule (i.e. any rule except the* Looping *rule) on a leaf α of $\mathcal{T}$. Let $\mathcal{I}$ be an interpretation. $\mathcal{I} \models \alpha$ iff there exists a child β of α in $\mathcal{T}'$ s.t. $\mathcal{I} \models \beta$.*

Proof. For *Propositional splitting* there are two branches β_1, β_2. If $\mathcal{I} \models \alpha$ then either $\mathcal{I}_p \models |P_{e_1,\ldots,e_k}|_{\rho_\mathcal{I}}$ or $\mathcal{I}_p \models |\neg P_{e_1,\ldots,e_k}|_{\rho_\mathcal{I}}$, and consequently either $\mathcal{I} \models \beta_1$ or $\mathcal{I} \models \beta_2$. Conversely it is easily seen that if we have $\mathcal{I} \models \beta_1$ or $\mathcal{I} \models \beta_2$ then $\mathcal{I} \models \alpha$.

Similarly we write β_1, β_2 for the two branches of *Constraint splitting*. By completeness of linear arithmetic, either $\models \exists \mathsf{i}(C\rho_\mathcal{I})$ or $\models \forall \mathsf{i} \neg(C\rho_\mathcal{I})$ for any interpretation $\mathcal{I}$ (notice that by the application condition of the rule, all variables occurring in C are parameters, thus i is the only free variable of $C\rho_\mathcal{I}$). Suppose $\mathcal{I} \models \alpha$, then in the first case $\mathcal{I} \models \beta_1$, in the second case the iteration is empty so $\mathcal{I} \models \beta_2$. Conversely if $\mathcal{I} \models \beta_1$ then it is trivial that $\mathcal{I} \models \alpha$ and if $\mathcal{I} \models \beta_2$ then $\models \forall \mathsf{i} \neg(C\rho_\mathcal{I})$ and thus $|\, \Delta_{\mathsf{i}|C}\, \pi|_{\rho_\mathcal{I}} = \varepsilon$ (following the notations of *Constraint splitting*), so $\mathcal{I} \models \alpha$.

We do not detail all the rewrite rules, which have only one conclusion. Suppose β is obtained from α by rewriting a schema S_1 into S_2. Let $\mathcal{I}$ be a model of α or β (whether one proves the "only if" or the "if" implication of the lemma). It is easily proved (using the side conditions of each rewrite) that $\mathcal{I}_p(|S_1|_{\rho_\mathcal{I}}) = \mathcal{I}_p(|S_2|_{\rho_\mathcal{I}})$, i.e. for any $\mathcal{I}$ the propositional realizations under $\rho_\mathcal{I}$ of S_1 and S_2 have the same value w.r.t. $\mathcal{I}_p$. Actually we even have that for all rules except *Expansion*, and for every environment ρ of S_1, $|S_1|_\rho$ and $|S_2|_\rho$ are equivalent.

Consider *Interval splitting*. Suppose we have $\mathcal{I} \models \alpha$ then either $(l.e_1 \lhd k.e_2)\rho_\mathcal{I}$ or $(l.e_1 \not\lhd k.e_2)\rho_\mathcal{I}$ is valid (by completeness of linear arithmetic). Furthermore, it is easily seen that for every i, $l.e_1 \lhd k.e_2$ and $k.\mathsf{i} \lhd e_1$ entail $l.\mathsf{i} \lhd e_2$ (as $k, l > 0$; one

has to carefully make the distinction between the cases $e_1 = 0$ and $e_1 \neq 0$), and $l.e_1 \not\lhd k.e_2$ and $l.\mathsf{i} \lhd e_2$ entail $k.\mathsf{i} \lhd e_1$. Consequently, in both cases removing the entailed constraint does not affect the propositional realization of the iteration, and thus $\mathcal{I} \models \beta_1$ or $\mathcal{I} \models \beta_2$. Now suppose $\mathcal{I} \models \beta_1$. Then $(l.e_1 \lhd k.e_2)\rho_\mathcal{I}$ is valid. Thus we have in the exact same way: $\forall \mathsf{i}(k.\mathsf{i} \lhd e_1 \Rightarrow l.\mathsf{i} \lhd e_1)\rho_\mathcal{I}$. And thus $(k.\mathsf{i} \lhd e_1)\rho_\mathcal{I}$ is equivalent to $(k.\mathsf{i} \lhd e_1 \wedge l.\mathsf{i} \lhd e_1)\rho_\mathcal{I}$. So $\mathcal{I} \models \alpha$. The case $\mathcal{I} \models \beta_2$ is similar. □

A leaf is *irreducible* iff no rule of DPLL* applies to it.

Lemma 2. *If a leaf α in $\mathcal{T}$ is irreducible and not closed then $\mathcal{T}$ is satisfiable.*

Proof. We first show that $S_\mathcal{T}(\alpha)$ does not contain iterations. If there are iterations then there are iterations which are not contained into any other iteration. Let $\Delta_{\mathsf{i}|C}\,\pi$ be such an iteration. $\Delta_{\mathsf{i}|C}\,\pi$ cannot be empty by irreducibility of *Constraint splitting* and *Emptiness*. So by irreducibility of *Interval splitting* and elimination of quantifiers in linear arithmetic, C can be restricted to a non-empty disjunction of inequalities $e_1 \leq \mathsf{i} \wedge \mathsf{i} \leq e_2$. Thus $C[e_1/\mathsf{i}]$ is valid and *Unfolding* can apply which is impossible. Hence there cannot be any iteration which is not contained into any other iteration, thus there cannot be any iteration at all. So $S_\mathcal{T}(\alpha)$ is constructed only with $\wedge, \vee, \neg$ and indexed propositions. Hence it is easily seen that any literal L s.t. $L \sqsubset S_\mathcal{T}(\alpha)$ satisfies $L \sqsubset_\Box S_\mathcal{T}(\alpha)$. Thus either $L \in \mathcal{L}_\mathcal{T}(\alpha)$ or $L^c \in \mathcal{L}_\mathcal{T}(\alpha)$ by irreducibility of *Propositional splitting*. As a consequence if there existed such a literal, *Expansion* and then *Algebraic simplification* would have applied, turning every occurrence of L into $\bot$ or $\top$. Hence $S_\mathcal{T}(\alpha)$ does not contain any literal, and by irreducibility of *Algebraic simplification* $S_\mathcal{T}(\alpha)$ is either $\bot$, impossible as the branch is not closed, or $\top$, which is satisfiable. Finally $\mathcal{L}$ cannot contain two contradictory literals, because the application conditions of *Propositional splitting* ensure that L is added to $\mathcal{L}_\mathcal{T}(\alpha)$ only if neither $L \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ nor $L^c \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$. We conclude with Lemma 1 that the initial tableau is satisfiable. □

Theorem 1 (Soundness). *Let $\mathcal{T}$ be a tableau. If a tableau $\mathcal{T}'$ is obtained from $\mathcal{T}$ by application of the extension rules, and if $\mathcal{T}'$ contains an irreducible and not closed leaf then $\mathcal{T}$ is satisfiable.*

Proof. This follows immediately from Lemmata 1 and 2. □

We now prove that the procedure is complete w.r.t. satisfiability i.e. that if S has a model then every sequence of tableaux constructed from S (in a fair way) eventually contains an irreducible and not closed branch. To do this we assume the existence of a model, then we define a well-founded measure w.r.t. this model and we show that it strictly decreases at each rule application (Lemma 3). Thus there will be a leaf s.t. this measure is minimal, so no rule can apply on it. We then use Lemma 1 to show that this leaf cannot be closed. This is formalized in the proof of Theorem 2.

Intuitively, we take a model $\mathcal{I}$ and apply DPLL* by focusing only on the branch for which $\mathcal{I}$ is a model (by Lemma 1, there always exists such a branch).

Then, in this branch, all iterations will progressively be unfolded. This process will stop because the iteration has a fixed length in $\mathcal{I}$. Concretely, the iterations will be unfolded rank by rank until it only remains an empty iteration which will then be removed either by *Constraint splitting* or *Emptiness*. Once this is done for all iterations, what remains is the propositional realization of the original schema w.r.t. $\rho_{\mathcal{I}}$ (except that in the meantime, some literals may have been evaluated, leading to some possible simplifications). So we have a propositional formula and *Algebraic simplification* applies until we obtain $\top$, i.e. a node which is irreducible and not closed.

As the reader will see, the presented measure is not trivial (in particular $m^3_{\mathcal{I}}$). We first outline the encountered problems that justify such a definition. From the explanations of the previous §, the measure must be greater when the schema contains an iteration (and the longer the iteration is, the greater shall be the measure). For instance such a measure would be strictly lower after an application of *Unfolding*. *Emptiness* divides an iteration into two iterations such that the sum of their lengths is equal to the length of the original iteration, thus the measure remains the same. This is easily circumvented, e.g. by squaring the length of iterations. A bigger problem occurs with *Expansion* which *adds* iterations where there was no iteration before. A natural solution is to define another measure that decreases on *Expansion*, e.g. the number of possible applications of the rule. Then we give this measure a higher priority (via a lexicographic ordering). But this does not work because *Unfolding* duplicates the pattern π (following the notations of the rule) and can thus increase the number of possible applications of *Expansion*. Similar problems are encountered with *Emptiness*: this rule also makes a kind of unfolding but, following the notations of the rule, $C \wedge \exists \mathsf{i}' C'$ can be unsatisfiable. In such a case, it means that the unfolding is fake, it just allows us to introduce the information $\forall \mathsf{i}' \neg C'$ in the rightmost iteration. So in this case we have just introduced a new iteration, without even decomposing the original one. Once again we could define another measure, e.g. the number of possible applications of *Emptiness*, but this is increased by *Unfolding*.

We now present formally our solution which requires the two following "auxiliary" functions:

$$\begin{aligned} \mu(x,0) &= (x+2)^2 & \nu(0) &= 1 \\ \mu(x,k+1) &= (\mu(x,k)+x+2)^2 & \nu(k+1) &= \mu(\nu(k),0)+1 \end{aligned}$$

The following results are easily proved (most of them by induction). They sum up all the properties of ν and μ that are useful to prove that the measure decreases.

Proposition 2.

1. $\forall x, k \in \mathbb{N},\ \mu(x,k) \geq 4$
2. $\forall x, k \in \mathbb{N},\ \mu(x,k) \geq x$
3. $\forall x, y, k \in \mathbb{N},\ x < y \Rightarrow \mu(x,k) < \mu(y,k)$
4. $\forall k_1, k_2 \in \mathbb{N},\ k_1 < k_2 \Rightarrow \nu(k_1) < \nu(k_2)$
5. $\forall x, y, k \in \mathbb{N}$ *s.t.* $y \geq 1,\ \mu(x,k) + y < \mu(x+y,k)$
6. $\forall x, y, k \in \mathbb{N},\ \mu(x,k) + \mu(y,k) < \mu(x+y,k+1)$

7. $\forall k \in \mathbb{N},\ \nu(k+1) > \mu(\nu(k), 0)$

We can now define the measure. Let $\mathcal{I}$ be an interpretation and α a node of a tableau $\mathcal{T}$. We set $m_{\mathcal{I}}(\alpha, \mathcal{T}) \stackrel{\text{def}}{=} (m^1_{\mathcal{I}}(\alpha), m^2_{\mathcal{I}}(\alpha), m^3_{\mathcal{I}}(S_{\mathcal{T}}(\alpha)), m^4(\alpha), m^5(\alpha))$ ordered using the lexicographic extension of the usual ordering on natural numbers, where $m^1_{\mathcal{I}}(\alpha)$, $m^2_{\mathcal{I}}(\alpha)$, $m^3_{\mathcal{I}}(S_{\mathcal{T}}(\alpha))$, $m^4(\alpha)$, $m^5(\alpha))$ are defined as follows:

1. For a parameter n of $S_{\mathcal{T}}(\alpha)$, $m^1_{\mathcal{I},\mathsf{n}}(\alpha) \stackrel{\text{def}}{=} \rho_{\mathcal{I}}(\mathsf{n})$. $m^1_{\mathcal{I}}(\alpha)$ is defined as the multiset extension of $m^1_{\mathcal{I},\mathsf{n}}(\alpha)$ to all parameters of $S_{\mathcal{T}}(\alpha)$.
2. $m^2_{\mathcal{I}}(\alpha)$ is the number of atoms (different from $\bot, \top$) that occur in $|S_{\mathcal{T}}(\alpha)|_{\rho_{\mathcal{I}}}$ but not in $|\bigwedge_{\mathcal{L}_{\mathcal{T}}(\alpha)}|_{\rho_{\mathcal{I}}}$.
3. $m^3_{\mathcal{I}}(S_{\mathcal{T}}(\alpha))$ is defined by induction on the structure of $\Pi_{S_{\mathcal{T}}(\alpha)}$:
   - $m^3_{\mathcal{I}}(\top) \stackrel{\text{def}}{=} m^3_{\mathcal{I}}(\bot) \stackrel{\text{def}}{=} 1$
   - $m^3_{\mathcal{I}}(\neg\pi) \stackrel{\text{def}}{=} m^3_{\mathcal{I}}(\pi) + 1$.
   - $m^3_{\mathcal{I}}(\pi_1 \vartriangle \pi_2) \stackrel{\text{def}}{=} m^3_{\mathcal{I}}(\pi_1) + m^3_{\mathcal{I}}(\pi_2)$
   - $m^3_{\mathcal{I}}(\Delta_{\mathsf{i}|C}\,\pi) \stackrel{\text{def}}{=} \mu(\sum_{i\in E} m^3_{\mathcal{I}}(\pi[i/\mathsf{i}]), n_{it})$ where E is the set $\{i \in \mathbb{Z} \mid C\rho_{\mathcal{I}}[i/\mathsf{i}] \text{ is valid}\}$ (E is finite since C encloses i) and n_{it} is the number of iterations $\nabla_{\mathsf{i}'|C'}\,\pi'$ occurring in π s.t. *Emptiness* can apply on $\Delta_{\mathsf{i}|C}\,\pi[\nabla_{\mathsf{i}'|C'}\,\pi']$ (with the notations of *Emptiness*).
   - $m^3_{\mathcal{I}}(P_{e_1,\ldots,e_k}) \stackrel{\text{def}}{=} \nu(n_l)$ where n_l is the number of literals $P_{f_1,\ldots,f_k} \in \mathcal{L}_{\mathcal{T}}(\alpha)$ s.t. *Expansion* applies on $P_{e_1,\ldots,e_k}$.
4. $m^4(\alpha)$ is the number of possible applications of *Interval splitting* on α.
5. $m^5(\alpha)$ is the number of iterations $\Delta_{\mathsf{i}|C}\,\pi$ of $S_{\mathcal{T}}(\alpha)$ s.t. $C_{S_{\mathcal{T}}(\alpha)} \wedge \forall \mathsf{i} \neg C$ is satisfiable.

An *extended child* of a node α is a child of α if α is not a bud node, or the companion node of α otherwise. For extended children we have the following weaker version of Lemma 1:

Proposition 3. *Let $\mathcal{T}$, $\mathcal{T}'$ be tableaux s.t. $\mathcal{T}'$ is obtained by applying any rule of* DPLL* *on a leaf α of $\mathcal{T}$. If α is satisfiable then there exists a satisfiable extended child of α in $\mathcal{T}'$.*

Proof. Indeed if α is a bud node then it follows from Definitions 7 and 8, otherwise it follows from Lemma 1. □

Let $\mathcal{I}$ be a model of α and β a satisfiable extended child of α, $\mathcal{I}^\alpha_\beta$ is defined as follows: if α is a bud node then there is $\mathcal{J}$ s.t. $\mathcal{J}(\mathsf{n}) < \mathcal{I}(\mathsf{n})$ for some parameter n and $\mathcal{J} \models \beta$. We set $\mathcal{I}^\alpha_\beta \stackrel{\text{def}}{=} \mathcal{J}$. If α is not a bud node then $\mathcal{I}^\alpha_\beta \stackrel{\text{def}}{=} \mathcal{I}$. We can now prove the main lemma, which states that the measure strictly decreases when applying a rule.

Lemma 3. *Let $\mathcal{T}$, $\mathcal{T}'$ be tableaux s.t. $\mathcal{T}'$ is deduced from $\mathcal{T}$ by applying a rule on a leaf α. If there is a model $\mathcal{I}$ of α then for every satisfiable extended child β of α in $\mathcal{T}'$ we have $m_{\mathcal{I}^\alpha_\beta}(\beta, \mathcal{T}') < m_{\mathcal{I}}(\alpha, \mathcal{T})$.*

Proof. By inspection of the extension rules:

Rule	$m^1_{\mathcal{I}}(\alpha)$	$m^2_{\mathcal{I}}(\alpha)$	$m^3_{\mathcal{I}}(S_{\mathcal{T}}(\alpha))$	$m^4(\alpha)$	$m^5(\alpha)$
Propositional splitting	$\leq$	$<$			
Constraint splitting	$\leq$	$\leq$	$\leq$	$\leq$	$<$
Algebraic simplification	$\leq$	$\leq$	$<$		
Unfolding	$\leq$	$\leq$	$<$		
Emptiness	$\leq$	$\leq$	$<$		
Expansion	$\leq$	$\leq$	$<$		
Interval splitting	$\leq$	$\leq$	$\leq$	$<$	
Looping	$<$				

$\leq$ (resp. $<$) means that the corresponding measure *does not increase* (resp. *strictly decreases*) by application of the rule.

- For $m^1_{\mathcal{I}}(\alpha)$, *Looping* strictly decreases by Definition 7. In all other cases, $\mathcal{I}^\alpha_\beta = \mathcal{I}$ so all parameters have the same values, thus $m^1_{\mathcal{I}}(\alpha)$ is constant.
- When *Propositional splitting* applies, either $P_{e_1,\dots,e_k} \sqsubset_\square S$ or $\neg P_{e_1,\dots,e_k} \sqsubset_\square S$ (following the notations of *Propositional splitting*), so either $|P_{e_1,\dots,e_k}|_{\rho_{\mathcal{I}}} \sqsubset |S_{\mathcal{T}}(\alpha)|_{\rho_{\mathcal{I}}}$ or $|\neg P_{e_1,\dots,e_k}|_{\rho_{\mathcal{I}}} \sqsubset |S_{\mathcal{T}}(\alpha)|_{\rho_{\mathcal{I}}}$. Hence either $P_{e_1,\dots,e_k}$ or $\neg P_{e_1,\dots,e_k}$ is added to $\mathcal{L}_{\mathcal{T}}(\alpha)$, thus $m^2_{\mathcal{I}}(\alpha)$ strictly decreases. It is obvious that other rules (except *Looping*) cannot increase the number of atoms in $|S_{\mathcal{T}}(\alpha)|_{\rho_{\mathcal{I}}}$ (even rules that duplicate a pattern, namely *Unfolding* or *Emptiness*: indeed the number of atoms is increased in the schema but its propositional realization remains the same, see the proof of Lemma 1) thus they cannot increase $m^2_{\mathcal{I}}(\alpha)$. *Looping* has already been shown to be decreasing so we do not mind that this rule possibly increases due to the lexicographic ordering. The same argument allows us to omit the *Propositional splitting* rule in the following, and similarly for the each subsequent measure.
- We detail the case of $m^3_{\mathcal{I}}(S_{\mathcal{I}}(\alpha))$, rule by rule (the notations used here — n_l, n_{it}, E — are the same as in the definition of $m^3_{\mathcal{I}}$):
  1. *Constraint splitting*: the pattern does not change but one must take care that Context($S_{\mathcal{T}}(\alpha)$) does change, so n_l and n_{it} may increase. However *Constraint splitting* only strengthens the context and so cannot increase those numbers.
  2. *Algebraic simplification*: obvious by inspection of all rules. The two first items of Proposition 2 enable to conclude for rules involving an iteration.
  3. *Unfolding*: the result follows from the fifth item of Proposition 2 (taking $y = m^3_{\mathcal{I}}(\pi[e/\mathsf{i}])$; $m^3_{\mathcal{I}}(\pi) \geq 1$ for every pattern π so indeed $y \geq 1$). Similarly to *Constraint splitting*, n_l and n_{it} cannot increase.
  4. For *Emptiness*, the result follows from the sixth item of Proposition 2: it is easily seen that n_{it} strictly decreases from the application conditions of *Emptiness* and because those conditions are not satisfied anymore after the rewrite. μ and n_{it} have been precisely defined to handle this rule.
  5. For *Expansion*, the result follows from the seventh item of Proposition 2: it is obvious that n_l strictly decreases during the rewrite. ν and n_l have been precisely defined to handle this rule.
  6. *Interval splitting* only changes the domain of an iteration. With a similar reasoning as in Lemma 1, one easily gets that in both branches the set

E remains the same. Furthermore, similarly to the *Constraint splitting* case, n_l and n_{it} cannot increase.
- The last two measures follow the conditions of the corresponding decreasing rules, easily entailing their corresponding behaviors. It is obvious that *Constraint splitting* cannot increase $m^4(\alpha)$. □

A *derivation* is a (possibly infinite) sequence of tableaux $(\mathcal{T}_i)_{i\in I}$ s.t. I is either $[0..k]$ for some $k \geq 0$, or $\mathbb{N}$ and s.t. for all $i > 0$, $\mathcal{T}_i$ is obtained from $\mathcal{T}_{i-1}$ by applying one of the rules. A derivation is *fair* iff either there is $i \in I$ s.t. $\mathcal{T}_i$ contains an irreducible, not closed, leaf or if for all $i \in I$ and every leaf α in $\mathcal{T}_i$ there is $j \geq i$ s.t. a rule is applied on α in $\mathcal{T}_j$ (i.e. no leaf can be indefinitely "freezed").

Theorem 2 (Model Completeness). *Let $\mathcal{T}_0$ be a satisfiable tableau. If $(\mathcal{T}_i)_{i\in I}$ is a fair derivation then there are $k \in I$ and a leaf α_k in $\mathcal{T}_k$ s.t. α_k is irreducible and not closed.*

Proof. By Lemma 1, for every model $\mathcal{I}$ of $\mathcal{T}_0$ and for all $k \in I$, $\mathcal{T}_k$ contains a leaf α_k s.t. $\mathcal{I} \models \alpha_k$. Consider such $\mathcal{I}, k, \alpha_k$ s.t. $m_{\mathcal{I}}(\alpha_k, \mathcal{T}_k)$ is minimal (exist since $m_{\mathcal{I}}(\alpha_k, \mathcal{T}_k)$ is well-founded). By Lemma 1, α_k is not closed. Suppose that α_k is not irreducible. Then, since the derivation is fair, there is $l > k$ s.t. a rule is applied on α_k in the tableau $\mathcal{T}_l$. By Proposition 3 there exists a satisfiable extended child β of α_k in $\mathcal{T}_l$ and $m_{\mathcal{I}^\alpha_\beta}(\beta, \mathcal{T}_l) < m_{\mathcal{I}}(\alpha_k, \mathcal{T}_k)$ by Lemma 3. This is impossible by minimality of $m_{\mathcal{I}}(\alpha_k, \mathcal{T}_k)$. Hence α_k is irreducible. □

4 Looping Refinements

The notion of loop introduced in Definition 8 is undecidable, thus, in practice, we use decidable refinements of looping.

Definition 10. *A binary relation between schemata is a* looping refinement *iff it is a subset of the looping relation.*

Termination proofs work by showing that the set of schemata which are generated by the procedure is finite up to some (decidable) looping refinement. We make precise this notion:

Definition 11. *Let $\mathcal{S}$ be a set of schemata and $\triangleright$ a looping refinement. A schema $[S] \in \mathcal{S}$ is a* $\triangleright$-maximal companion *(or just* maximal companion *when $\triangleright$ is obvious from the context) w.r.t. $\mathcal{S}$ iff there is no $S' \in \mathcal{S}$ s.t. $[S] \triangleright S'$. The set of all $\triangleright$-maximal companions w.r.t. $\mathcal{S}$ is written $\mathcal{S}/\triangleright$. If $\mathcal{S}/\triangleright$ is finite then we say that $\mathcal{S}$ is* finite up to $\triangleright$.

Notice that we use the notations $[S]$ and $\mathcal{S}/\triangleright$ as if we were talking of an equivalence class and a quotient set *but $\triangleright$ is generally not an equivalence.* However the underlying intuition is often very close and we think that using this notation makes it easier to understand the proofs, as soon as the reader is clearly aware that this is not an equivalence relation.

4.1 Equality up to a Shift

We now present perhaps the simplest refinement of looping. A *shiftable* is a schema, a linear constraint, a pattern, a linear expression or a tuple of those. The refinement is defined on shiftables (and not only on schemata) in order to handle those objects in a uniform way. This is useful in the termination proof of Section 5.

Definition 12. *Let s, s' be shiftables and n a variable. If $s' = s[\mathsf{n} - k/\mathsf{n}]$ for some $k > 0$, then s' is* equal to s up to a shift of k on n*, written $s' \rightrightarrows^{\mathsf{n}} s$ (or $s' \rightrightarrows^{\mathsf{n}}_k s$ when we want to make k explicit).*

Notice that we use a *syntactical* equality e.g. we do not care about associativity or commutativity of $\wedge$ and $\vee$ when the shiftables are schemata, nor do we use linear constraint equivalence when the shiftables are linear constraints. This makes this refinement less powerful but trivial to implement and easier to reason with.

Proposition 4. *Let n be a variable, the restriction of $\rightrightarrows^{\mathsf{n}}$ to schemata having n as a parameter is a looping refinement.*

Proof. Let S_1, S_2 be schemata s.t. $S_1 \rightrightarrows^{\mathsf{n}}_k S_2$ for some $k > 0$. Let $\mathcal{I}$ be a model of S_1. We define $\mathcal{J}$ s.t. $\rho_{\mathcal{J}}(\mathsf{n}) \stackrel{\text{def}}{=} \rho_{\mathcal{I}}(\mathsf{n}) - k$, $\rho_{\mathcal{J}}(\mathsf{m}) \stackrel{\text{def}}{=} \rho_{\mathcal{I}}(\mathsf{m})$ for $\mathsf{m} \neq \mathsf{n}$ and $\mathcal{J}_p \stackrel{\text{def}}{=} \mathcal{I}_p$. It is obvious that $|S_1|_{\rho_{\mathcal{I}}} = |S_2|_{\rho_{\mathcal{J}}}$ and as $\mathcal{J}_p = \mathcal{I}_p$, $\mathcal{J} \models S_2$. It is also obvious that $\rho_{\mathcal{J}}(\mathsf{n}) < \rho_{\mathcal{I}}(\mathsf{n})$. □

Proposition 5. *For all shiftables s, s_1, s_2, if $s_1 \rightrightarrows^{\mathsf{n}} s$ and $s_2 \rightrightarrows^{\mathsf{n}} s$ then either $s_1 \rightrightarrows^{\mathsf{n}} s_2$ or $s_2 \rightrightarrows^{\mathsf{n}} s_1$ or $s_1 = s_2$.*

Finally $\rightrightarrows^{\mathsf{n}}$ is transitive but neither reflexive (e.g. $P_{\mathsf{n}} \not\rightrightarrows^{\mathsf{n}} P_{\mathsf{n}}$), nor irreflexive (e.g. $P_1 \rightrightarrows^{\mathsf{n}} P_1$). It is irreflexive for shiftables containing n, and reflexive for shiftables not containing n (in which case equality up to a shift just amounts to equality).

Definition 13. *A set of shiftables $\mathcal{S}$ s.t. all its different elements are comparable w.r.t. $\rightrightarrows^{\mathsf{n}}$ is called a* looping chain *. We extend the notion of maximal companion to shiftables: a shiftable s is a* maximal companion *w.r.t. a set of shiftables $\mathcal{S}$ iff there is no $s' \in \mathcal{S}$ s.t. $s \rightrightarrows^{\mathsf{n}} s'$. If a looping chain $\mathcal{S}$ contains a shiftable s which is a maximal companion w.r.t. $\mathcal{S}$ then $\mathcal{S}$ is a* well-founded chain*.*

From the previous remarks a looping chain has the form: $\cdots \rightrightarrows^{\mathsf{n}} s_{i-1} \rightrightarrows^{\mathsf{n}} s_i \rightrightarrows^{\mathsf{n}} s_{i+1} \rightrightarrows^{\mathsf{n}} \cdots$, hence justifying the name "looping chain". Then, by considering all its totally comparable subsets, any set of schemata can be seen as a union of looping chains. A well-founded chain has the form $\cdots \rightrightarrows^{\mathsf{n}} s_2 \rightrightarrows^{\mathsf{n}} s_1 \rightrightarrows^{\mathsf{n}} s_0$, where s_0 is a maximal companion w.r.t. the chain.

We focus now on sets which are *finite up to equality up to a shift*, in short "$\rightrightarrows^{\mathsf{n}}$-finite" (i.e. sets which are finite unions of well-founded chains): termination proofs go by showing that the set of all schemata possibly generated by DPLL* is $\rightrightarrows^{\mathsf{n}}$-finite, thus ensuring that the *Looping* rule will eventually apply. To prove such results we need to reason by induction on the structure of a schema. To do this properly we need closure properties for $\rightrightarrows^{\mathsf{n}}$-finite sets i.e. if we know

that two sets are $\rightrightarrows^{\mathsf{n}}$-finite, we would like to be able to combine them and preserve the $\rightrightarrows^{\mathsf{n}}$-finite property. This is generally not possible, e.g. for two $\rightrightarrows^{\mathsf{n}}$-finite sets of shiftables $\mathcal{S}_1$ and $\mathcal{S}_2$, the set $\mathcal{S}_1 \times \mathcal{S}_2$ (remember that shiftables are closed by tuple construction) is generally *not* $\rightrightarrows^{\mathsf{n}}$-finite. For instance take $\mathcal{S}_1 = \{P_{\mathsf{n}}, P_{\mathsf{n}-1}, P_{\mathsf{n}-2}, , \dots\}$ ($\mathcal{S}_1$ is $\rightrightarrows^{\mathsf{n}}$-finite with $\mathcal{S}_1/\rightrightarrows^{\mathsf{n}} = \{P_{\mathsf{n}}\}$) and $\mathcal{S}_2 = \{P_{\mathsf{n}}\}$ (which is finite and thus $\rightrightarrows^{\mathsf{n}}$-finite). Then $\{(P_{\mathsf{n}}, P_{\mathsf{n}}), (P_{\mathsf{n}-1}, P_{\mathsf{n}}), (P_{\mathsf{n}-2}, P_{\mathsf{n}}), \dots\}$ is not $\rightrightarrows^{\mathsf{n}}$-finite: indeed for every $i \in \mathbb{N}$, $(P_{\mathsf{n}-i}, P_{\mathsf{n}})$ is a maximal companion in $\mathcal{S}_1 \times \mathcal{S}_2$, there is thus an infinite set of maximal companions. Consequently $\mathcal{S}_1 \times \mathcal{S}_2$ is not $\rightrightarrows^{\mathsf{n}}$-finite. This example also shows that $\rightrightarrows^{\mathsf{n}}$-finite sets are not even closed by cartesian product with a finite set. Hence we have to restrict our closure operators.

Definition 14. *Let* n *be a variable. A shiftable* s *is* translated *w.r.t.* n *iff for every linear expression* e *occurring in* s *and containing* n *there is* $k \in \mathbb{Z}$ *s.t.* $e = \mathsf{n}+k$ *(i.e. neither* $k.\mathsf{n}$ *nor* $\mathsf{n}+\mathsf{i}$ *are allowed, where* $k \in \mathbb{Z}$, $k \neq 0$ *and* $\mathsf{i} \in \mathcal{IV}$*).*

Assume that s *is translated w.r.t.* n*. The* deviation *of* s *w.r.t.* n*, written* $\delta(s)$*, is defined as* $\delta(s) \stackrel{def}{=} \max\{k_1 - k_2 \mid k_1, k_2 \in \mathbb{Z}, \mathsf{n}+k_1, \mathsf{n}+k_2 \text{ occur in } s\}$. $\delta(s) \stackrel{def}{=} 0$ *if* s *does not contain* n*. Let* $k \in \mathbb{N}$*, we write* $\mathfrak{B}_k$ *for the set* $\{s \mid \delta(s) \leq k\}$.

Theorem 3. *Let* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *be two sets of shiftables translated w.r.t. a variable* n*. If* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are* $\rightrightarrows^{\mathsf{n}}$*-finite then, for any* $k \in \mathbb{N}$*, the set* $\mathcal{S}_1 \times \mathcal{S}_2 \cap \mathfrak{B}_k$*, written* $\mathcal{S}_1 \times_k \mathcal{S}_2$*, is* $\rightrightarrows^{\mathsf{n}}$*-finite.*

One can notice that, in the counter-example given before Definition 14, the deviations of schemata in $\mathcal{S}_1 \times \mathcal{S}_2$ are unbounded.

Proof. We construct a bijective function $f : \mathcal{S}_1/\rightrightarrows^{\mathsf{n}} \times \mathcal{S}_2/\rightrightarrows^{\mathsf{n}} \times [-k..k] \to \mathcal{S}_1 \times_k \mathcal{S}_2/\rightrightarrows^{\mathsf{n}}$: as $\mathcal{S}_1/\rightrightarrows^{\mathsf{n}}$, $\mathcal{S}_2/\rightrightarrows^{\mathsf{n}}$ and $[-k..k]$ are finite, $\mathcal{S}_1 \times_k \mathcal{S}_2/\rightrightarrows^{\mathsf{n}}$ is finite, hence the result. Informally f associates to each pair of maximal companions a maximal companion in $\mathcal{S}_1 \times_k \mathcal{S}_2$, however there are as many new maximal companions as there are possible deviations (actually twice as many), hence the dependency on $[-k..k]$. Let $[s_1] \in \mathcal{S}_1/\rightrightarrows^{\mathsf{n}}$, $[s_2] \in \mathcal{S}_2/\rightrightarrows^{\mathsf{n}}$ and $d \in [-k..k]$, we now construct $f([s_1], [s_2], d)$.

First of all if $[s_1]$ or $[s_2]$ does not contain n then $f([s_1], [s_2], d) \stackrel{\text{def}}{=} ([s_1], [s_2])$ independently of d. Then for any pair $(s_1, s_2) \in \mathcal{S}_1 \times_k \mathcal{S}_2$ s.t. s_1 or s_2 does not contain n, it is easily seen that $(s_1, s_2) \rightrightarrows^{\mathsf{n}} ([s_1], [s_2])$ (we let the reader observe that this would not necessarily be the case if both s_1 and s_2 contained n). Furthermore $([s_1], [s_2])$ is a maximal companion. Indeed suppose that there is another (s_1', s_2') s.t. $([s_1], [s_2]) \rightrightarrows^{\mathsf{n}} (s_1', s_2')$ then necessarily $[s_1] \rightrightarrows^{\mathsf{n}} s_1'$ which contradicts the fact that $[s_1]$ is a maximal companion w.r.t. $\mathcal{S}_1$.

So from now on we assume that both $[s_1]$ and $[s_2]$ contain n. Hence every shiftable s s.t. $s \rightrightarrows^{\mathsf{n}} s_1$ or $s \rightrightarrows^{\mathsf{n}} s_2$ also contains n. As a consequence, for every shiftable s, $\max(s) \stackrel{\text{def}}{=} \max\{k \in \mathbb{Z} \mid \mathsf{n}+k \text{ occurs in } s\}$ is well-defined.

We first prove that $\{(s_1, s_2) \mid s_1 \rightrightarrows^{\mathsf{n}} [s_1], s_2 \rightrightarrows^{\mathsf{n}} [s_2], \max(s_1) - \max(s_2) = d\}$ is a looping chain. Thus we prove that for all $s_1, s_1' \in \mathcal{S}_1$ and $s_2, s_2' \in \mathcal{S}_2$ s.t. $s_1 \rightrightarrows^{\mathsf{n}} [s_1]$, $s_1' \rightrightarrows^{\mathsf{n}} [s_1]$, $s_2 \rightrightarrows^{\mathsf{n}} [s_2]$, $s_2' \rightrightarrows^{\mathsf{n}} [s_2]$, $\max(s_1) - \max(s_2) = d$ and $\max(s_1') - \max(s_2') = d$ there is $k > 0$ s.t. either $(s_1', s_2') \rightrightarrows^{\mathsf{n}}_k (s_1, s_2)$ or

$(s_1, s_2) \rightrightarrows^{\mathsf{n}}_k (s'_1, s'_2)$ or $(s_1, s_2) = (s'_1, s'_2)$. By Proposition 5 we have either $s_1 \rightrightarrows^{\mathsf{n}}_k s'_1$, $s'_1 \rightrightarrows^{\mathsf{n}}_k s_1$ or $s_1 = s'_1$ for some $k > 0$ and either $s_2 \rightrightarrows^{\mathsf{n}}_{k'} s'_2$, $s'_2 \rightrightarrows^{\mathsf{n}}_{k'} s_2$ or $s_2 = s'_2$ for some $k' > 0$. Suppose $s'_1 \rightrightarrows^{\mathsf{n}}_k s_1$ then $\max(s_1) - \max(s'_1) = k$. From $\max(s_1) - \max(s_2) = d$ and $\max(s'_1) - \max(s'_2) = d$ it easily follows that $\max(s_2) - \max(s'_2) = k$. As $k > 0$, this entails that we cannot have $s_2 \rightrightarrows^{\mathsf{n}} s'_2$ or $s_2 = s'_2$. Hence the only possibility is $s'_2 \rightrightarrows^{\mathsf{n}} s_2$, and more precisely $s'_2 \rightrightarrows^{\mathsf{n}}_{k'} s_2$ with $k' = k$. As a consequence $(s'_1, s'_2) \rightrightarrows^{\mathsf{n}}_k (s_1, s_2)$. The case $s_1 \rightrightarrows^{\mathsf{n}}_k s'_1$ is symmetric, and the case $s_1 = s'_1$ easily entails $s_2 = s'_2$ by taking $k = 0$ in the previous equations.

Then we prove that it is a well-founded chain. Notice that if $(s_1, s_2) \rightrightarrows^{\mathsf{n}} (s'_1, s'_2)$ then $s_1 \rightrightarrows^{\mathsf{n}} s'_1$. So if a looping chain $\cdots \rightrightarrows^{\mathsf{n}} (s_1^{i-1}, s_2^{i-1}) \rightrightarrows^{\mathsf{n}} (s_1^{i}, s_2^{i}) \rightrightarrows^{\mathsf{n}} (s_1^{i+1}, s_2^{i+1}) \rightrightarrows^{\mathsf{n}} \cdots$ does not contain a maximal companion then one of the looping chains $\cdots \rightrightarrows^{\mathsf{n}} s_1^{i-1} \rightrightarrows^{\mathsf{n}} s_1^{i} \rightrightarrows^{\mathsf{n}} s_1^{i+1} \rightrightarrows^{\mathsf{n}} \cdots$ or $\cdots \rightrightarrows^{\mathsf{n}} s_2^{i-1} \rightrightarrows^{\mathsf{n}} s_2^{i} \rightrightarrows^{\mathsf{n}} s_2^{i+1} \rightrightarrows^{\mathsf{n}} \cdots$ does not contain a maximal companion, either. By hypothesis this is false in our case. As a consequence there is indeed a maximal companion for the looping chain $\{(s_1, s_2) \mid s_1 \rightrightarrows^{\mathsf{n}} [s_1], s_2 \rightrightarrows^{\mathsf{n}} [s_2], \max(s_1) - \max(s_2) = d\}$, we set $f([s_1], [s_2], d)$ to be this maximal companion. It is now trivial that $(\mathcal{S}_1 \times_k \mathcal{S}_2)/\rightrightarrows^{\mathsf{n}} = f[\mathcal{S}_1/\rightrightarrows^{\mathsf{n}} \times \mathcal{S}_2/\rightrightarrows^{\mathsf{n}} \times [-k..k]]$: for any pair $(s_1, s_2) \in \mathcal{S}_1 \times_k \mathcal{S}_2$, $(s_1, s_2) \rightrightarrows^{\mathsf{n}} f([s_1], [s_2], \max(s_1) - \max(s_2))$ and $f([s_1], [s_2], \max(s_1) - \max(s_2))$ is a maximal companion w.r.t. $\mathcal{S}_1 \times_k \mathcal{S}_2$. Notice that $|\max(s_1) - \max(s_2)| \leq k$ because $(s_1, s_2) \in \mathfrak{B}_k$. □

As trivial corollaries we get (where all the involved shiftables are translated w.r.t. n):

- $\{S_1 \vartriangle S_2 \mid S_1 \in \mathcal{S}_1, S_2 \in \mathcal{S}_2\} \cap \mathfrak{B}_k$, where $\vartriangle \in \{\wedge, \vee\}$, is $\rightrightarrows^{\mathsf{n}}$-finite when $\mathcal{S}_1$ and $\mathcal{S}_2$ are $\rightrightarrows^{\mathsf{n}}$-finite.
- $\{(\bigwedge_{i|C} \Pi_S) \wedge C_S \mid S \in \mathcal{S}, C \in \mathcal{C}\} \cap \mathfrak{B}_k$ is $\rightrightarrows^{\mathsf{n}}$-finite when $\mathcal{S}$ and $\mathcal{C}$ are $\rightrightarrows^{\mathsf{n}}$-finite .
- $\{e_1 = e_2 \mid e_1 \in \mathcal{E}_1, e_2 \in \mathcal{E}_2\} \cap \mathfrak{B}_k$ is $\rightrightarrows^{\mathsf{n}}$-finite when $\mathcal{E}_1$ and $\mathcal{E}_2$ are sets of linear expressions, $\rightrightarrows^{\mathsf{n}}$-finite (this corollary will be useful in the proof of Lemma 8, which explains why equality up to a shift is defined on shiftables and not only on schemata).

4.2 Refinement Extensions

Equality up to a shift is generally not powerful enough to detect cycles, so we now define simple extensions that allow better detection. Consider for example the schema S defined in Example 2. Using DPLL* there is a branch which contains: $S' \stackrel{\text{def}}{=} P_1 \wedge \bigwedge_{\mathsf{i}=1}^{\mathsf{n}-1}(P_{\mathsf{i}} \Rightarrow P_{\mathsf{i}+1}) \wedge \neg P_{\mathsf{n}} \wedge \neg P_{\mathsf{n}+1} \wedge \mathsf{n} \geq 0 \wedge \mathsf{n} - 1 \geq 0$. S' loops on S but S' is not equal to S up to a shift. However $\neg P_{\mathsf{n}+1}$ is pure in S' (i.e. $P_{\mathsf{n}+1} \not\sqsubseteq S'$) so $\neg P_{\mathsf{n}+1}$ may be evaluated to true. Therefore we obtain $P_1 \wedge \bigwedge_{\mathsf{i}=1}^{\mathsf{n}-1}(P_{\mathsf{i}} \Rightarrow P_{\mathsf{i}+1}) \wedge \neg P_{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{n} - 1 \geq 0$, i.e. $S[\mathsf{n} - 1/\mathsf{n}] \wedge \mathsf{n} \geq 0$. But $\mathsf{n} - 1 \geq 0$ entails $\mathsf{n} \geq 0$ so we can remove $\mathsf{n} \geq 0$ and finally get $S[\mathsf{n} - 1/\mathsf{n}]$.

We now generalise this example, thereby introducing two new looping refinements: the *pure literal* extension and the *redundant constraint* extension (both

of them actually take an existing looping refinement and extend it into a more powerful one, hence the name "extension"). We could have defined them as rules rather than looping refinements, however this way the results can be useful not only to DPLL* but also to any other system working with iterated schemata e.g. they are applicable without any modification to the system STAB defined in [2].

Pure Literals. As usual a literal L is *(propositionally) pure* in a formula ϕ iff its complement does not occur positively in ϕ. The pure literal rule is standard in propositional theorem proving: it consists in evaluating a literal L to true in a formula ϕ if L is pure in ϕ. It is well-known that this operation preserves satisfiability but it is now often omitted as looking for occurrences of a literal generally costs more than the benefits of its removal. In our case, however, dropping this optimization frequently results in non termination.

The notion of pure literal has to be adapted to schemata. The conditions on L must be strengthened in order to take iterations into account. For instance, if $L = P_{\mathsf{n}}$ and $S = \bigvee_{\mathsf{i}=1}^{2\mathsf{n}} \neg P_{\mathsf{i}}$ then L is not pure in S since $\neg P_{\mathsf{i}}$ is the complement of L for $\mathsf{i} = \mathsf{n}$ (and $1 \leq \mathsf{n} \leq 2\mathsf{n}$). On the other hand $P_{2\mathsf{n}+1}$ is pure in S (since $2\mathsf{n}+1 \notin [1..2\mathsf{n}]$). It is actually easy to see that $\sqsubset_\diamond$ is the right tool to formalize this notion.

Definition 15. *A literal L is* pure *in a schema S iff for every environment ρ of S, $|L|_\rho$ is propositionally pure in $|S|_\rho$.*

It is easily seen that L is pure in S iff $L^c \not\sqsubset_\diamond S$, thus by decidability of $\sqsubset_\diamond$, it is decidable to determine if a literal is pure or not.

The substitution of an indexed proposition $P_{e_1,\ldots,e_k}$ by a pattern π' in a pattern π, written $\pi[\pi'/P_{e_1,\ldots,e_k}]$, is defined as follows:

$$P_{e_1,\ldots,e_k}[\pi'/P_{e_1,\ldots,e_k}] \stackrel{\text{def}}{=} \pi'$$

$$Q_{f_1,\ldots,f_k}[\pi'/P_{e_1,\ldots,e_k}] \stackrel{\text{def}}{=} Q_{f_1,\ldots,f_k} \qquad \text{if } P \neq Q \text{ or } f_i \neq e_i \text{ for some } i \in [1..k]$$

$$(\pi_1 \vartriangle \pi_2)[\pi'/P_{e_1,\ldots,e_k}] \stackrel{\text{def}}{=} \pi_1[\pi'/P_{e_1,\ldots,e_k}] \vartriangle \pi_2[\pi'/P_{e_1,\ldots,e_k}] \qquad (\vartriangle \in \{\vee, \wedge\})$$

$$(\underset{\mathsf{i}|C}{\Delta}\, \pi)[\pi'/P_{e_1,\ldots,e_k}] \stackrel{\text{def}}{=} \underset{\mathsf{i}|C}{\Delta}\, \pi[\pi'/P_{e_1,\ldots,e_k}] \qquad (\Delta \in \{\bigvee, \bigwedge\})$$

Notice that this is a trivial syntactic substitution, e.g. $(\neg P_1 \wedge \bigvee_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}})[\top/P_1] = \neg\top \wedge \bigvee_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}}$ and not $\neg\top \wedge (\top \vee \bigvee_{\mathsf{i}=2}^{\mathsf{n}} P_{\mathsf{i}})$. Actually the latter would be a mistake because we do not know whether $\mathsf{n} \geq 1$ or not. The definition naturally extends to a schema S with $S[\pi'/P_{e_1,\ldots,e_k}] \stackrel{\text{def}}{=} \Pi_S[\pi'/P_{e_1,\ldots,e_k}]$.

Proposition 6. *Let L be a literal pure in a schema S. If S has a model $\mathcal{I}$ then $S[\top/L]$ has a model $\mathcal{J}$ s.t. $\rho_{\mathcal{I}}(\mathsf{n}) = \rho_{\mathcal{J}}(\mathsf{n})$ for every parameter n of S.*

Conversely if $S[\top/L]$ has a model $\mathcal{I}$ then S has a model $\mathcal{J}$ s.t. $\rho_{\mathcal{I}}(\mathsf{n}) = \rho_{\mathcal{J}}(\mathsf{n})$ for every parameter n of S.

Proof. Let $\mathcal{I}$ be a model of S. $|S|_{\rho_{\mathcal{I}}}$ is thus satisfiable. As L is pure in S, $|L|_{\rho_{\mathcal{I}}}$ is pure in $|S|_{\rho_{\mathcal{I}}}$ (and thus in $|S[\top/L]|_{\rho_{\mathcal{I}}}$). So by the classical result that

the satisfiability of a propositional formula is preserved when removing a pure literal, $|S|_{\rho_\mathcal{I}}[\top/|L|_{\rho_\mathcal{I}}]$ is satisfiable. As $|S|_{\rho_\mathcal{I}}[\top/|L|_{\rho_\mathcal{I}}] = |S[\top/L]|_{\rho_\mathcal{I}}[\top/|L|_{\rho_\mathcal{I}}]$, $|S[\top/L]|_{\rho_\mathcal{I}}$ is also satisfiable. We define $\mathcal{J}_p$ as one of its models and $\rho_\mathcal{J}$ as $\rho_\mathcal{I}$. $\mathcal{J}$ is obviously a model of $S[\top/L]$ and indeed $\rho_\mathcal{I}(\mathsf{n}) = \rho_\mathcal{J}(\mathsf{n})$ for every parameter n of S. The proof of the converse is symmetric. □

A schema S in which all pure literals have been substituted with $\top$ is written purified(S).

Definition 16. *Let $\triangleright$ be a looping refinement. We call* the pure extension *of $\triangleright$ the relation $\triangleright'$: $S_1 \triangleright' S_2 \Leftrightarrow \text{purified}(S_1) \triangleright \text{purified}(S_2)$.*

Proposition 7. *The pure extension of a looping refinement is a looping refinement.*

Proof. Consider S_1, S_2 s.t. $S_1 \triangleright' S_2$, i.e. $\text{purified}(S_1) \triangleright \text{purified}(S_2)$. Let $\mathcal{I}$ be a model of S_1. By Proposition 6, there exists a model $\mathcal{I}'$ of purified(S_1) s.t. $\rho_{\mathcal{I}'}(\mathsf{n}) = \rho_\mathcal{I}(\mathsf{n})$ for every parameter n of S_1. Then, as $\triangleright$ is a looping refinement and by Definition 7, there is a model $\mathcal{J}'$ of purified(S_2) s.t. $\rho_{\mathcal{J}'}(\mathsf{n}) < \rho_{\mathcal{I}'}(\mathsf{n})$ for some parameter n of purified(S_1) (and thus of S_1) and $\rho_{\mathcal{J}'}(\mathsf{n}) \leq \rho_{\mathcal{I}'}(\mathsf{n})$ for other parameters of S_1. Then by Proposition 6, there exists a model $\mathcal{J}$ of S_2 s.t. $\rho_{\mathcal{J}'}(\mathsf{n}) = \rho_\mathcal{J}(\mathsf{n})$ for every parameter n of S_2. From a model $\mathcal{I}$ of S_1, we constructed a model $\mathcal{J}$ of S_2 s.t. $\rho_\mathcal{J}(\mathsf{n}) < \rho_\mathcal{I}(\mathsf{n})$ for some parameter n of S_1 and $\rho_\mathcal{J}(\mathsf{n}) \leq \rho_\mathcal{I}(\mathsf{n})$ for other parameters, i.e. we proved that S_1 loops on S_2. □

Redundant Constraints. This extension is justified by the fact that DPLL* often leads to constraints of the form $\mathsf{n} > 0$, then $\mathsf{n} > 0 \wedge \mathsf{n} - 1 > 0$, then $\mathsf{n} > 0 \wedge \mathsf{n} - 1 > 0 \wedge \mathsf{n} - 2 > 0$, etc. Such constraints contain redundant information, which can be an obstacle to the detection of cycles in a proof.

Definition 17. *Any normal form of a schema S by the following rewrite rules:*

$$\begin{array}{cc} C_1 \wedge \cdots \wedge C_k \to C_1 \wedge \cdots \wedge C_{k-1} & \text{if } \{C_1, \ldots, C_{k-1}\} \models C_k \\ C \to \bot & \text{if } C \text{ is unsatisfiable} \end{array}$$

is called a constraint-irreducible *schema of S.*

By decidability of satisfiability in linear arithmetic, it is easy to compute a constraint-irreducible schema of S.

Definition 18. *Let $\triangleright$ be a looping refinement. We call* the constraint-irreducible extension *of $\triangleright$ the relation $\triangleright'$ s.t. for all S_1, S_2, $S_1 \triangleright' S_2$ iff there exists S_1' (resp. S_2') a constraint-irreducible schema of S_1 (resp. S_2) s.t. $S_1' \triangleright S_2'$.*

Proposition 8. *The constraint-irreducible extension of a looping refinement is a looping refinement.*

Proof. It is easy to show that if S (resp. a constraint-irreducible of S) has a model $\mathcal{I}$ then any constraint-irreducible of S (resp. S) has a model $\mathcal{J}$ s.t. $\rho_\mathcal{I}(\mathsf{n}) = \rho_\mathcal{J}(\mathsf{n})$ for every parameter n of S. Then the proof goes exactly the same way as in the proof of Proposition 7. □

Generalisation. We can generalize Propositions 7 and 8:

Proposition 9. *Let $\triangleright$ be a looping refinement, and $\star$ a binary relation among schemata s.t. for all schemata S_1, S_2 if $S_1 \star S_2$ then the satisfiability of S_1 is equivalent to the satisfiability of S_2, preserving the values of the parameters. The relation $\triangleright'$ s.t. for all S_1, S_2, $S_1 \triangleright' S_2$ iff there exists S'_1, S'_2 s.t. $S_1 \star S'_1$, $S_2 \star S'_2$ and $S'_1 \triangleright S'_2$, is a looping refinement.*

And we can generalize Definitions 16 and 18: $\triangleright'$ is called the $\star$-*extension* of $\triangleright$.

Of course this construction has an interest only if $\triangleright'$ catches more looping cases than $\triangleright$. It can be seen as working with normal forms of schemata w.r.t. $\star$ which can be better suited to $\triangleright$ than their non-normal counterparts. From the two previous definitions and from the requirement that satisfiability "fits well" with $\star$, it can be observed that extensions would be seen in some other context as just *optimisations* (see e.g. the pure literal rule, or the remark about normal forms). In the context of schemata, those are generally more than just optimizations as they may be required for *termination*. Interestingly enough circumscribing those extensions to the looping rule allows us to keep a high-level description of the main proof system and a modular presentation of looping.

5 Decidable Classes

We now present some classes of schemata for which DPLL* terminates.

5.1 Regularly Nested Schemata

Definition 19 (Regularly Nested Schema). *An iteration $\Delta_{\mathsf{i}|C}\,\pi$ is* framed *iff there are two expressions e_1, e_2 s.t. $C \Leftrightarrow e_1 \leq \mathsf{i} \wedge \mathsf{i} \leq e_2$. $[e_1..e_2]$ is called the* frame *of the iteration.*

A schema S is:

- Monadic *iff all indexed propositions occurring in S have only one index.*
- Framed *iff all iterations occurring in it are framed.*
- Aligned on $[e_1..e_2]$ *iff it is framed and all iterations have the same frame $[e_1..e_2]$.*
- Translated *iff it is translated w.r.t. every variable occurring in it.*
- Regularly Nested *iff it has a unique parameter n, it is monadic, translated and aligned on $[k..\mathsf{n}-l]$ for some $k, l \in \mathbb{Z}$.*

The definitions extend to a node α of a tableau $\mathcal{T}$ by considering its schema $S_{\mathcal{T}}(\alpha)$.

Notice that regularly nested schemata allow the nesting of iterations. But they are too weak to express the binary multiplier presented in the Introduction (since only monadic propositions are considered).

Example 5. $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \bigvee_{\mathsf{j}=1}^{\mathsf{n}} (P_{\mathsf{i}} \Rightarrow Q_{\mathsf{j}}) \wedge \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \neg Q_{\mathsf{i}} \wedge \bigvee_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}}$ is regularly nested.

We divide *Constraint splitting* into two disjoint rules: *framed-Constraint splitting* (resp. *non framed-Constraint splitting*) denotes *Constraint splitting* with the restriction that $\Delta_{\mathsf{i}|C}\,\pi$ (following the notations of the rule) is framed (resp. not framed). We consider the following strategy $\mathfrak{S}$ for applying the extension rules on a regularly nested schema:

1. First only framed-*Constraint splitting* applies until irreducibility.
2. Then all other rules except *Unfolding* apply until irreducibility with the restriction that *Expansion* rewrites P_{e_1} iff e_1 contains no variable other than the parameter of the schema (notice that there is only one index because the schema is monadic).
3. Finally only *Unfolding* applies until irreducibility, with the restriction that if the unfolded iteration is framed then e (in the definition of *Unfolding*) is the upper bound of the frame. We then go back to 1.

For the *Looping* rule we use equality up to a shift with its pure and constraint-irreducible extensions (it is trivial that the order in which the extensions are done does not matter). It is easy to prove that $\mathfrak{S}$ preserves completeness.

Interval splitting and *Emptiness* never apply when the input schema is regularly nested. Indeed let $\Delta_{\mathsf{i}|C}\,\pi$ be an iteration of the schema. C cannot contain an expression of the form $k.e$, hence *Interval splitting* cannot apply. No variable other than i or the parameter can be free in C (due to the frame of the form $[k..\mathsf{n}-l]$), thus *Emptiness* cannot apply. However *Expansion* may introduce non framed iterations, but no variable other than i or the parameter can be free in C because *Expansion* only applies if e_1 contains no variable other than the parameter of the schema. All this shall become clear in the next section.

5.2 Termination of DPLL* for Regularly Nested Schemata

The proof that $\mathfrak{S}$ terminates for regularly nested schemata goes by showing that the set $\{S\mathcal{L}_\mathcal{T}(\alpha) \mid \alpha \text{ is a node of } \mathcal{T}\}$ — i.e. the set of schemata generated all along the procedure — is (roughly[4]) finite up to the constraint-irreducible and pure extensions of equality up to a shift. As $S\mathcal{L}_\mathcal{T}(\alpha) = \Pi_{S_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)} \wedge \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)}$, this set is equal to $\{\Pi_{S_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)} \wedge \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \mid \alpha \text{ is a node of } \mathcal{T}\}$. So the task can approximately be divided into four: prove that the set of patterns is finite up to a shift (Lemma 9), prove that the set of constraints is finite up to a shift (Lemma 8), prove that the set of partial interpretations is finite up to a shift (Lemma 7, Corollary 3) and combine the three results thanks to Theorem 3 (Corollary 4).

Tracing DPLL*. Among those tasks, the hardest is the first one, because it requires an induction on the structure of $\Pi_{S_\mathcal{T}(\alpha)}$. For this induction to be achieved properly we need to "trace" the evolution under $\mathfrak{S}$ of every subpattern of $\Pi_{S_\mathcal{T}(\alpha)}$. A subpattern can be uniquely identified by its position. So we extend DPLL*

[4] This set will actually be restricted to *alignment nodes*, see Definition 21.

into T-DPLL* (for **T**raced DPLL*), by adding to the pair $(S_\mathcal{T}(\alpha), \mathcal{L}_\mathcal{T}(\alpha))$ labelling nodes in DPLL* a third component containing a set of positions of $\Pi_{S_\mathcal{T}(\alpha)}$. Along the execution of the procedure, this subpattern may be moved, duplicated, deleted, some context may be added around it, some of its subpatterns may be modified. Despite all those modifications, we are able to follow the subpattern thanks to the set of positions in the labels.

As usual, a position is a finite sequence of natural numbers, ϵ denotes the empty sequence, $s_1.s_2$ denotes the concatenation of s_1 and s_2 and $\leq$ denotes the prefix ordering. The *positions* of a pattern π are defined as follows: ϵ is a position in π; if p is a position in π then $1.p$ is a position in $\neg\pi$, $\bigwedge_{\mathrm{i}|C}\pi$ and $\bigvee_{\mathrm{i}|C}\pi$; let $i \in \{1,2\}$, if p is a position in π_i then $i.p$ is a position in $\pi_1 \vee \pi_2$ and $\pi_1 \wedge \pi_2$.

For two sequences s_1, s_2 s.t. s_2 is a prefix of s_1, $s_2 \setminus s_1$ is the sequence s.t. $s_2.(s_2 \setminus s_1) = s_1$. In particular for two positions p_1, p_2 s.t. p_2 is a prefix of p_1, $p_2 \setminus p_1$ can be seen as the position *relatively to* p_2 of the subterm in position p_1 in S.

Definition 20 (T-DPLL*). *A* T-DPLL* tableau $\mathcal{T}$ *is the same as a* DPLL* *tableau except that a node α is labeled with a triple $(S_\mathcal{T}(\alpha), \mathcal{L}_\mathcal{T}(\alpha), \mathcal{P}_\mathcal{T}(\alpha))$ where $\mathcal{P}_\mathcal{T}(\alpha)$ is a set of positions in $\Pi_{S_\mathcal{T}(\alpha)}$.* T-DPLL* *keeps the behavior of* DPLL* *for $S_\mathcal{T}(\alpha)$ and $\mathcal{L}_\mathcal{T}(\alpha)$, we only describe the additional behavior for $\mathcal{P}_\mathcal{T}(\alpha)$ as follows: $p \to p_1, \ldots, p_k$ means that p is deleted and $p_1, \ldots, p_k$ are added to $\mathcal{P}_\mathcal{T}(\alpha)$.*

- *Splitting rules and the* Expansion *rewrite rule leave $\mathcal{P}_\mathcal{T}(\alpha)$ as is.*
- *Rewrite rules. We write q for the position of the subpattern of Π_S which is rewritten. We omit* Emptiness *as it never applies.*
  - Algebraic simplification. *For $p > q$:*

$p \to q.(1 \setminus (q \setminus p))$	*for rules where π occurs on both sides of the rewrite (following the notations of Definition 6), and if p is the position of a subpattern of π*
$p \to \emptyset$	*otherwise*

  - Unfolding.

$$\text{for } p > q: \quad p \to\ q.1.(q \setminus p),\ q.2.1.(q \setminus p)$$

Let α, β be nodes of a T-DPLL**-tableau $\mathcal{T}$ s.t. $\beta \prec \alpha$. For two patterns π_1, π_2, we write $\pi_1 \rightsquigarrow^\beta_\mathcal{T} \pi_2$ iff $\pi_1 = \Pi_{S_\mathcal{T}(\alpha)}|_{p_1}$ and $\pi_2 = \Pi_{S_\mathcal{T}(\beta)}|_{p_2}$ for some positions $p_1 \in \mathcal{P}_\mathcal{T}(\alpha)$ and $p_2 \in \mathcal{P}_\mathcal{T}(\beta)$.*

Notice that $\mathfrak{S}$ is naturally extended to T-DPLL* tableaux.

The *stripped* of a T-DPLL* tableau is the tree obtained by removing the last component (i.e. the set of positions) of each of its nodes' label. The following proposition is trivial:

Proposition 10. *(i) If $\mathcal{T}$ is a* T-DPLL* *tableau then its stripped is a* DPLL* *tableau. (ii) Conversely if $\mathcal{T}$ is a* DPLL* *tableau of root $(S, \emptyset, \top)$, and p is a position in Π_S, then there is a unique* T-DPLL* *tableau $\mathcal{T}_p$ of root $(S, \emptyset, \top, \{p\})$, s.t. the stripped of $\mathcal{T}_p$ is equal to $\mathcal{T}$.*

$\mathcal{T}_p$ is called the *decorated* of $\mathcal{T}$ w.r.t. p.

Alignment Nodes. The set $\{S\mathcal{L}_\mathcal{T}(\alpha) \mid \alpha \text{ is a node of } \mathcal{T}\}$ is actually *not* finite up to a shift. We have to restrict ourselves to a particular kind of nodes, called *alignment nodes*. Eventually, $\{S\mathcal{L}_\mathcal{T}(\alpha) \mid \alpha \text{ is an } \textit{alignment} \text{ node of } \mathcal{T}\}$ will indeed be finite up to a shift.

From now on, $\mathcal{T}$ is a T-DPLL* tableau whose root schema is regularly nested of parameter n and of alignment $[k..\mathsf{n}-l]$ for some $k, l \in \mathbb{Z}$.

Definition 21 (Alignment Node). *A node of $\mathcal{T}$ is an* alignment node *iff it is irreducible by step 2 of $\mathfrak{S}$ (see page 24).*

Proposition 11. *Let α, β be nodes of $\mathcal{T}$ s.t. β is obtained by applying step 3 on α. (i) Every iteration that occurs in $S_\mathcal{T}(\beta)$ occurs in $S_\mathcal{T}(\alpha)$. (ii) Furthermore if α is aligned on $[e_1..e_2]$ for some expressions e_1, e_2, then either $C_{S_\mathcal{T}(\beta)} \models e_1 > e_2$ or $C_{S_\mathcal{T}(\beta)} \models e_1 \leq e_2$.*

Proof. (i) is trivial as only *Constraint splitting* can apply. It applies only if $C_{S_\mathcal{T}(\alpha)} \wedge \forall \mathsf{i} \neg C$ is satisfiable (following the notations of the rule). If it is not the case then we have immediately $C_{S_\mathcal{T}(\alpha)} \models e_1 \leq e_2$, hence (ii). Otherwise *Constraint splitting* can apply and (ii) is obvious. □

Proposition 12. *Let α, β be nodes of $\mathcal{T}$ s.t. β is obtained by applying step 2 on α. If an iteration $\Delta_{\mathsf{i}|C}\,\pi$ occurs in $S_\mathcal{T}(\beta)$ then there is π' s.t. $\Delta_{\mathsf{i}|C}\,\pi'$ occurs in $S_\mathcal{T}(\alpha)$.*

Proof. Either $\Delta_{\mathsf{i}|C}\,\pi$ comes from the rewrite of π into π' by rules of step 2 (in which case the result is obvious), or it is new and has been introduced by the rules. We show that the latter case is actually impossible. By observing the conclusion of each rule that can apply in step 2, only *Expansion* can introduce new iterations (as *Emptiness* and *Interval splitting* cannot apply), so suppose that $\Delta_{\mathsf{i}|C}\,\pi$ was introduced by *Expansion*. By definition of $\mathfrak{S}$, C must have the form: $\delta.\mathsf{n} + k_1 \neq \mathsf{n} + k_2 \wedge \mathsf{i} = 0$ where $\delta \in \{0, 1\}$, $k_1, k_2 \in \mathbb{N}$ (and n is the only parameter of the schema). But then (non framed) *Constraint splitting* must have applied on $\Delta_{\mathsf{i}|C}\,\pi$ (it can indeed apply because if the condition of application was not fulfilled, then the domain of the iteration would be valid, and *Algebraic simplification* would have removed it). $\Delta_{\mathsf{i}|C}\,\pi$ is removed in the right branch of *Constraint splitting*, so we focus on the left branch: due to the added constraint, $\text{Context}(S_1) \Rightarrow \exists \mathsf{i} C$ (following the notations of *Algebraic simplification*) is valid. Furthermore, as i was a fresh variable when *Expansion* applied, π does not contain i. Thus *Algebraic simplification* must have applied and removed the iteration. Consequently $\Delta_{\mathsf{i}|C}\,\pi$ cannot have been introduced by *Expansion*. □

Proposition 13. *Let α, β be nodes of $\mathcal{T}$ s.t. β is obtained by applying step 3 on α. If α is aligned on $[e_1..e_2]$, and $C_{S_\mathcal{T}(\alpha)} \models e_1 \leq e_2$, then β is aligned on $[e_1..e_2 - q]$, for some $q > 0$.*

Proof. As $C_{S_\mathcal{T}(\alpha)} \models e_1 \leq e_2$, *Unfolding* can apply, and thus turn all the frames into $[e_1..e_2 - 1]$. Notice that we may also have $C_{S_\mathcal{T}(\alpha)} \models e_1 \leq e_2 - q$ for some $q > 0$, in which case *Unfolding* can apply q times more per iteration. □

Lemma 4. *An alignment node α of $\mathcal{T}$ is aligned on $[k..\mathsf{n}-l-j]$ for some $j \in \mathbb{N}$. Furthermore if an alignment node $\beta \prec \alpha$ is aligned on $[k..\mathsf{n}-l-j']$ for some $j' \in \mathbb{N}$, then $j' > j$.*

Proof. The result is proved by induction on the number of alignment nodes above α. The base case follows from the fact that the root of $\mathcal{T}$ is regularly nested and thus aligned on $[k..\mathsf{n}-l]$. By Propositions 11 (i) and 12 applying step 1 and then step 2 preserves the alignment. Let α' be an alignment node s.t. $\alpha \prec \alpha'$, and there is no alignment node between α and α'. By induction α' is aligned on $[k..\mathsf{n}-l-j]$ for some $j \in \mathbb{N}$. Because α' is an alignment node, *Constraint splitting* must have applied between α' and α. Thus we have either $C_{S_\mathcal{T}(\alpha)} \models k > \mathsf{n}-l-j$ or $C_{S_\mathcal{T}(\alpha)} \models k \leq \mathsf{n}-l-j$, by Proposition 11 (ii). In the first case there are no more iterations and every subsequent node is trivially aligned. In the second case, by Proposition 13, every node after step 3 is aligned on $[k..\mathsf{n}-l-j']$ for some $j' > j$. Then, once again, by Propositions 11 (i) and 12, applying step 1 and step 2 preserves the alignment, so every next alignment node has the expected alignment. □

When an alignment node α of $\mathcal{T}$ is aligned on $[k..\mathsf{n}-l-j]$ for some $j \in \mathbb{N}$, we call α a *j-alignment node.*

Corollary 1. *Every alignment node of $\mathcal{T}$ is regularly nested.*

Proof. We have to check that no new parameter is introduced, that the schema is still monadic, still translated and still aligned on $[k..\mathsf{n}-l]$ for some $k, l \in \mathbb{Z}$. The alignment is an obvious consequence of Lemma 4. The "monadicity" is trivially preserved. The only way a new parameter could be introduced is when a connective binding a variable is removed. But it is easily seen that each rule which removes such a connective also removes the pattern in which the variable is bound, so no bound variable can become free. Finally the schema remains translated because a new arithmetic expression can only be introduced in DPLL* via an instantiation in *Unfolding* (or *Interval splitting* with $l.e_1$ and $k.e_2$, but it cannot apply). As a regularly nested schema is translated w.r.t. every variable, every expression occurring in it is either an integer or has the form $\mathsf{i}+k$ where i is a variable and $k \in \mathbb{Z}$, Instantiating a variable in an integer of course does not change the integer. Instantiating i in $\mathsf{i}+k$ with an integer turns the expression into another integer. Instantiating i in $\mathsf{i}+k$ with another expression $\mathsf{i}'+k'$, turns the expression into $\mathsf{i}'+k'+k$, which preserves the form of the expression. Hence in all cases translated property of the schema is preserved. □

Lemma 5. *Let $\mathcal{T}$ be a tableau whose root schema is regularly nested of parameter n. For every alignment node α of $\mathcal{T}$, n only occurs in the domains of iterations.*

Proof. We have to show that indices of all literals do not contain n. Suppose that $S_\mathcal{T}(\alpha)$ contains a literal L whose index contains n. We first show that we have either $L \sqsubset_\square \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ or $L^c \sqsubset_\square \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$. Indeed suppose it is not the case. We show that *Propositional splitting* can apply, i.e. that $L \sqsubset_\square S_\mathcal{T}(\alpha)$ or

$L^c \sqsubset_\Box S_\mathcal{T}(\alpha)$ (1), and neither $L \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ nor $L^c \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ (2):

1. Notice that this is not because L occurs in $S_\mathcal{T}(\alpha)$ that $L^c \sqsubset_\Box S_\mathcal{T}(\alpha)$ or $L \sqsubset_\Box S_\mathcal{T}(\alpha)$: indeed if L occurs in an iteration, there can be an environment where this iteration is empty, so L does not necessarily occur in the corresponding propositional realization. But as α is an alignment node, *Constraint splitting* has applied in step 1, adding the constraint that either all iterations were empty, or no iteration was empty. In the first case, no iteration remains (because *Algebraic simplification* must have applied in step 2) so L necessarily occurs outside an iteration, and thus $L^c \sqsubset_\Box S_\mathcal{T}(\alpha)$ or $L \sqsubset_\Box S_\mathcal{T}(\alpha)$. In the second case, we know by Proposition 12, that if the non-emptiness of iterations was true before step 2, then it is also true after step 2, i.e. at α. So we have indeed $L^c \sqsubset_\Box S_\mathcal{T}(\alpha)$ or $L \sqsubset_\Box S_\mathcal{T}(\alpha)$, and *Propositional splitting* indeed applies.
2. Suppose we have either $L \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ or $L^c \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$. As we supposed that neither $L \sqsubset_\Box \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ nor $L^c \sqsubset_\Box \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$, this means that there exists a literal $L' \in \mathcal{L}_\mathcal{T}(\alpha)$ satisfying the property ($\star$) that it has the same propositional symbol as L, not the same index in general, but this index may be the same in some environments (e.g. $L = P_\mathsf{n}$ and $\mathcal{L}_\mathcal{T}(\alpha) = \{P_1\}$). Then, as $L' \in \mathcal{L}_\mathcal{T}(\alpha)$, *Expansion* has necessarily applied on L by stating the disequality of the indices of L and L'. However it cannot be valid that those indices are the same, as this would entail $L \sqsubset_\Box \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ or $L^c \sqsubset_\Box \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$. So the disequality necessarily holds. This is easily seen that it is possible for one L', but it is not possible *for all* literals in $\mathcal{L}_\mathcal{T}(\alpha)$ satisfying ($\star$). Indeed this would contradict the assumption that $L \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ or $L^c \sqsubset_\Diamond \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$. This can be done formally by an induction on the number of literals satisfying ($\star$). So if this was not possible then the iteration would have been turned into its neutral element by *Algebraic simplification*, and so every occurrence of L would have been removed. This contradicts the initial assumption on L.

So we suppose that *Propositional splitting* has applied. Now, by definition of $\mathfrak{S}$, every occurrence of L found in $S_\mathcal{T}(\alpha)$ satisfies the conditions for the application of *Expansion* in $\mathfrak{S}$ (as the node is translated, an index cannot contain two distinct variables). As $L \sqsubset_\Box \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$ or $L^c \sqsubset_\Box \bigwedge_{\mathcal{L}_\mathcal{T}(\alpha)} \wedge C_{S_\mathcal{T}(\alpha)}$, there are $L_1, \ldots, L_q \in \mathcal{L}_\mathcal{T}(\alpha)$ of indices $e_1, \ldots, e_q$ s.t. all of them have the same propositional symbol as L, and $C_{S_\mathcal{T}(\alpha)} \Rightarrow \bigvee_{i \in 1..q} e = e_i$ is valid, where e is the index of L. Thus *Expansion* must have applied on L with all those literals, introducing iterations stating $e \neq e_1, \ldots, e \neq e_q$. The outermost iteration has thus necessarily be removed by *Algebraic simplification* and L must also have been removed before we reach step 3. □

Lemma 6. *Each of the steps 1, 2 and 3 terminates.*

Proof.

– Step 1: as already seen, framed-*Constraint splitting* applies at most once.
– Step 2: *Propositional splitting* can add new literals to the set of literals of a node. However this is done finitely many times, as it is easily seen that there are finitely many literals L s.t. $L \sqsubset_\square S$ or $L^c \sqsubset_\square S$. For each atom P_{e_1} s.t. e_1 contains no variable other than the parameter of the schema, *Expansion* applies as many times as there are literals with proposition symbol P in the set of literals. We just saw that this last number cannot grow infinitely, and the number of atoms in S cannot increase because *Unfolding* is not allowed in step 2. Finally, non-framed *Constraint splitting* applies as many times as there are non-framed iterations which is precisely the number of times where *Expansion* can apply.
– Step 3: only *Unfolding* can apply. This terminates because there are finitely many iterations in a schema, and because if e_1, e_2 are expressions, no constraint can entail $e_1 \leq e_2 - q$ for every $q \geq 0$. Notice that if *Constraint splitting* could apply in the meantime it would not terminate because constraints could be modified and thus there could be infinitely many e s.t. $\text{Context}(S_1) \Rightarrow C[e/\mathsf{i}]$ is valid (following the notations of *Unfolding*). □

Corollary 2. *Let b be a branch of $\mathcal{T}$ containing a node α then either b is finite or it contains an alignment node $\beta \prec \alpha$, i.e. an alignment node is always reached.*

Main Proof. The unfolding rules of DPLL* may introduce infinitely many distinct literals, e.g. from $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} P_{\mathsf{i}}$ we generate $P_{\mathsf{n}}, P_{\mathsf{n}-1}, \ldots$. In principle this obviously prevents termination, but the key point is that (as shown by Lemma 7) these literals will eventually become pure, which ensures that they will not be taken into account by the looping rule.

Definition 22. *Let S be a regularly nested schema. Let $A(S)$ be the set $\{q \in \mathbb{Z} \mid q$ is the index of a literal in $S\}$, we write $\min_{base}(S)$ for $\min(A(S))$ and $\max_{base}(S)$ for $\max(A(S))$.*

Let $B(S)$ be the set $\{q \in \mathbb{Z} \mid \mathsf{i}{+}q$ is the index of a literal in an iteration of $S\}$ (it is a subset of $\mathbb{Z}$, by limited progression). We write $\min_{ind}(S) = \min(B(S))$ and $\max_{ind}(S) = \max(B(S))$.

Proposition 14. *Let α, β be nodes of $\mathcal{T}$ s.t. $\beta \prec \alpha$. Then every literal of $S_{\mathcal{T}}(\beta)$ whose index is an integer occurs in $S_{\mathcal{T}}(\alpha)$. Any literal occurring in any node and whose index is an integer, occurs in the root schema S of $\mathcal{T}$. Consequently its index belongs to $[\min_{base}(S) .. \max_{base}(S)]$.*

Proof. First it easily seen that if a literal occurs after application of any rule *other than Unfolding*, then it already occurred before the application of the rule. This is not the case with *Unfolding* which can introduce a new literal, due to the substitution in its conclusion. Due to the restriction of *Unfolding* in $\mathfrak{S}$, this substitution replaces a variable with the last rank of an iteration. Furthermore *Unfolding* only applies on alignment nodes. By Lemma 4, it is known that such nodes are aligned and that the last rank of their iterations depends on the parameter. Hence every literal that is introduced by substituting a variable with

this last rank, cannot have an integer as index. So if a literal whose index is an integer occurs after application of any rule (including *Unfolding*), then it already occurred before the application of the rule.

Finally by induction on the length of the derivation, it is obvious that any literal occurring in any node and whose index is an integer, occurs in the root schema of $\mathcal{T}$. □

For the sake of simplicity we assume that *Propositional splitting* only applies on $P_{e_1,\ldots,e_k}$ if $P_{e_1,\ldots,e_k}$ occurs in $S_\mathcal{T}(\alpha)$ (notice that we can have $P_{e_1,\ldots,e_k} \sqsubset_\Box S_\mathcal{T}(\alpha)$ without $P_{e_1,\ldots,e_k}$ occurring in $S_\mathcal{T}(\alpha)$, e.g. $P_1 \sqsubset_\Box \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} P_\mathsf{i} \wedge \mathsf{n} \geq 1$). This simplifies much some technical details, and it can be proved that this is not restrictive.

Lemma 7. *Let S be the root schema of $\mathcal{T}$. There is $j_0 \in \mathbb{N}$ s.t. for every j-alignment node α of $\mathcal{T}$, if $j \geq j_0$ then every literal in $\mathcal{L}_\mathcal{T}(\alpha)$ of index $\mathsf{n}+q$ where $q < \min_{ind}(S) - l - j$ or $q > \max_{ind}(S) - l - j$ is pure in $S_\mathcal{T}(\alpha)$.*

Proof. Let $L \in \mathcal{L}_\mathcal{T}(\alpha)$. L is pure in $S_\mathcal{T}(\alpha)$ iff $L^c \not\sqsubset_\Diamond S_\mathcal{T}(\alpha)$, i.e. iff $\exists \mathsf{n}(C_{S_\mathcal{T}(\alpha)} \wedge \phi_{L^c}(S_\mathcal{T}(\alpha)))$ (where $\phi_{L^c}(S_\mathcal{T}(\alpha))$ is defined just before Proposition 1) does *not* hold, by Proposition 1. It is easily seen that, in our case (for the sake of simplicity we assume $L = P_{\mathsf{n}+q}$, the case $\neg P_{\mathsf{n}+q}$ is similar):

$$\begin{aligned}\phi_L(S_\mathcal{T}(\alpha)) = &\bigvee\{\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - j \wedge \mathsf{n} + q = \mathsf{i} + q') \mid \neg P_{\mathsf{i}+q'} \sqsubset S_\mathcal{T}(\alpha)\}\\ &\vee \bigvee\{\mathsf{n} + q = q' \mid \neg P_{q'} \sqsubset S_\mathcal{T}(\alpha)\}\\ &\vee \bigvee\{\mathsf{n} + q = \mathsf{n} + q' \mid \neg P_{\mathsf{n}+q'} \sqsubset S_\mathcal{T}(\alpha)\}\end{aligned}$$

But as α is an alignment node, if there were literals $\neg P_{q'} \sqsubset S_\mathcal{T}(\alpha)$ (resp. $\neg P_{\mathsf{n}+q'} \sqsubset S_\mathcal{T}(\alpha)$), then *Expansion* would have applied. Thus either such literals would have been eliminated or the corresponding constraint $\mathsf{n} + q = q'$ (resp. $\mathsf{n} + q = \mathsf{n} + q'$) would not hold in $C_{S_\mathcal{T}(\alpha)}$. So it only remains to prove that the following does not hold:

$$\exists \mathsf{n} \left(C_{S_\mathcal{T}(\alpha)} \wedge \bigvee\{\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - j \wedge \mathsf{n} + q = \mathsf{i} + q') \mid \neg P_{\mathsf{i}+q'} \sqsubset S_\mathcal{T}(\alpha)\}\right)$$

This amounts to:

$$\exists \mathsf{n} \left(C_{S_\mathcal{T}(\alpha)} \wedge \bigvee\{k + q' \leq \mathsf{n} + q \wedge \mathsf{n} + q \leq \mathsf{n} - l - j + q' \mid \neg P_{\mathsf{i}+q'} \sqsubset S_\mathcal{T}(\alpha)\}\right)$$

For every q' s.t. $\neg P_{\mathsf{i}+q'} \sqsubset S_\mathcal{T}(\alpha)$, we have $q' \leq \max_{ind}(S)$, by definition of $\max_{ind}(S)$. So if $q > \max_{ind}(S) - l - j$, then the above formula does not hold and we get the result.

Now if $q < \min_{ind}(S) - l - j$ then L is not pure in general, however we can find $j_0 \in \mathbb{N}$ s.t. if $j \geq j_0$ then it is actually impossible to have $q < \min_{ind}(S) - l - j$. We show that literals s.t. $q < \min_{ind}(S) - l - j$ can only be literals of the root schema S, so once all of them are pure, no other literal s.t. $q < \min_{ind}(S) - l - j$ will be introduced. Therefore we take j_0 to be the minimal j s.t. $\mathsf{n} + q > \mathsf{n} + l - j + \max_{ind}(S)$. First notice that, as L has been introduced in $\mathcal{L}_\mathcal{T}(\beta)$

by *Propositional splitting* at some node β, and thanks to the restriction made on *Propositional splitting* just before the lemma, L was occurring in $S_{\mathcal{T}}(\beta)$. Now either this literal was already occurring in the root schema or it has been introduced by an *Unfolding*. As α is aligned on $k..\mathsf{n}-l-j$, all literals that have been introduced so far by *Unfolding* have an index of the form $\mathsf{n}-l-j'+q'$ where $j' < j$ and $q' \in B(S)$ (see Definition 22 for the definition of $B(S)$). As $\min_{ind}(S) = \min(B(S))$ and $q < \min_{ind}(S) - l - j$, L cannot have been introduced by *Unfolding*. Thus L is indeed a literal of the root schema. Hence we can take j_0 as above (informally, iterations will be unfolded until all literals of the root schema are pure, when this is done we have our j_0). □

Corollary 3. *Let S be a regularly nested schema of parameter n and $\mathcal{T}$ a tableau of root schema S, then $\left\{\bigwedge_{\mathcal{L}_{\mathcal{T}}(\alpha)} \mid \alpha \text{ is an alignment node}\right\}$ is finite up to the pure extension of equality up to a shift on n.*

Proof. It amounts to prove that $\mathcal{L} := \bigwedge\{L \in \mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \text{ is an alignment node}, L \text{ is not pure in } S_{\mathcal{T}}(\alpha) \wedge \bigwedge_{\mathcal{L}_{\mathcal{T}}(\alpha)}\}$ is finite up to a shift on n. For every proposition symbol P and every $q \in [\min_{ind}(S_{\mathcal{T}}(\alpha)).. \max_{ind}(S_{\mathcal{T}}(\alpha))]$, we define the set $C(q,P) := \{P_{\mathsf{n}-l-j+q} \in \mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \text{ is a } j\text{-alignment node}, j \geq j_0\}$. $D(q,P)$ denotes the same set with $\neg P_{\mathsf{n}-l-j+q}$. E is the set of literals that occurred before a j-alignment node with $j \leq j_0$. Finally $F := \{P_q \in \mathcal{L}_{\mathcal{T}}(\alpha) \mid q \in \mathbb{Z}, \alpha \text{ is a } j\text{-alignment node}, j \geq j_0\}$. It is clear that:

$$\mathcal{L} = \bigcup_{q,P} C(q,P) \cup \bigcup_{q,P} D(q,P) \cup E \cup F$$

$C(q,P)$ and $D(q,P)$ are clearly finite up to a shift on n. As there are finitely many P and q, so are the sets $\bigcup_{q,P} C(q,P)$ and $\bigcup_{q,P} D(q,P)$. E is finite. Finally F is finite because all its elements are literals of the root schema S thanks to Proposition 14. Consequently $\mathcal{L}$ is indeed finite up to a shift. □

Lemma 8. *Let S be a regularly nested schema of parameter n and $\mathcal{T}$ a tableau of root schema S, then $\left\{C_{S_{\mathcal{T}}(\alpha)} \middle| \alpha \text{ is an alignment node}\right\}$ is finite up to the constraint-irreducible extension of equality up to a shift on n.*

Proof. As *Interval splitting* never applies, the only rule that introduces constraints is *Constraint splitting*. For a framed-*Constraint splitting*, the only constraints that may be introduced in an alignment node are of the form $\forall \mathsf{i} \neg(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n}-l-j)$ or $\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n}-l-j)$, for some $j \in \mathbb{N}$. Non-framed *Constraint splitting* introduces only constraints that come from the emptiness of an iteration added by *Expansion*. Thus those constraints have the form $e \star f$ where $\star \in \{=, \neq\}$, e comes from a literal in $S_{\mathcal{T}}(\alpha)$ and f comes from a literal in $\mathcal{L}_{\mathcal{T}}(\alpha)$. Thus if we are in a j-alignment node and e contains n then e belongs to the set $[\mathsf{n}-l-j+\min_{ind}(S)..\mathsf{n}-l-j+\max_{ind}(S)]$ by Lemma 7; if e does not contain n then it belongs to the set $[\min_{base}(S).. \max_{base}(S)]$ by Proposition 14; and f belongs to the set $[\min_{base}(S).. \max_{base}(S)] \cup [\mathsf{n}-l-j+\min_{ind}(S)..\mathsf{n}-l+\max_{ind}(S)]$.

We now prove that the set of added constraints is finite up to the constraint-irreducible extension of equality up to a shift. We distinguish various cases depending on the shape of the introduced constraints. Finally, we will combine those results thanks to Theorem 3.

- Framed constraint $\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - j)$: the set of generated constraints of this form is:

  $\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l)$
  $\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l) \wedge \exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - 1)$
  $\exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l) \wedge \exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - 1) \wedge \exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - 2)$
  etc.

  but we can remove the redundant constraints and obtain:

  $$\begin{array}{l} \exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l) \\ \exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - 1) \\ \exists \mathsf{i}(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - 2) \\ \text{etc.} \end{array}$$

  which is trivially $\rightrightarrows^{\mathsf{n}}$-finite.
- Framed constraint $\forall \mathsf{i} \neg(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - j)$: Once this constraint is added, there are no more iterations in the schema, so no other constraint of this form will be added. Thus the set of all constraints of this form that may be added in all the nodes is $\{\forall \mathsf{i} \neg(k \leq \mathsf{i} \wedge \mathsf{i} \leq \mathsf{n} - l - j) \mid j \in \mathbb{N}\}$ which is obviously finite up to a shift.
- Non-framed constraint with $e \in [\mathsf{n} - l - j + \min_{ind}(S)..\mathsf{n} - l - j + \max_{ind}(S)]$ and $f \in [\mathsf{n} - l - j + \min_{ind}(S)..\mathsf{n} - l + \max_{ind}(S)]$: then $e \star f$ is either valid or unsatisfiable. If it is valid then it is of course redundant so we do not even need to consider it. If it is unsatisfiable then, by constraint-irreducibility, we can consider that it is $\bot$. When an unsatisfiable constraint is added, the branch is closed, so no other constraint may be added. Thus the set of such constraints generated in this case is just $\{\bot\}$, trivially finite.
- Non-framed constraint with $e \in [\min_{base}(S).. \max_{base}(S)]$ and $f \in [\mathsf{n} - l - j + \min_{ind}(S)..\mathsf{n} - l - j - k + \max_{base}(S)]$, i.e. the considered set of constraints is:

  $$A \stackrel{\text{def}}{=} \left\{ e \star f \,\middle|\, \begin{array}{l} e \in [\min_{base}(S).. \max_{base}(S)] \\ f \in [\mathsf{n} - l - j + \min_{ind}(S)..\mathsf{n} - l - j - k + \max_{base}(S)] \\ j \in \mathbb{N} \end{array} \right\}$$

  It is a finite union of sets of the form $\{\mathsf{n} - j + q \mid j \in \mathbb{N}\}$ where $q \in \mathbb{Z}$. All such sets are $\rightrightarrows^{\mathsf{n}}$-finite, so A is $\rightrightarrows^{\mathsf{n}}$-finite. Then $[\min_{base}(S).. \max_{base}(S)]$ is obviously $\rightrightarrows^{\mathsf{n}}$-finite. so we get the result by the third corollary of Theorem 3 (with deviation 0 as no expression in $[\min_{base}(S).. \max_{base}(S)]$ contains n). Notice that the full interval on which f ranges ($[\mathsf{n} - l - j + \min_{ind}(S)..\mathsf{n} - l + \max_{ind}(S)]$) has been split on purpose, so that A can indeed be a finite union of $\rightrightarrows^{\mathsf{n}}$-finite sets.

– Non-framed constraint with $e \in [\min_{base}(S)..\max_{base}(S)]$ and $f \in [\mathsf{n} - l - j - k + \max_{base}(S) + 1..\mathsf{n} - l + \max_{ind}(S)]$, when $\star$ is $=$: this constraint states $e = f$. However we know that $k \leq \mathsf{n} - l - j$, so $\mathsf{n} \geq k + l + j$. As $e = f$, we have $\mathsf{n} = \mathsf{n} + e - f$. So $\mathsf{n} + e - f \geq k + l + j$, thus $f \leq \mathsf{n} + e - k - l - j$. As $e \leq \max_{base}(S)$, we obtain $f \leq \mathsf{n} + \max_{base}(S) - k - l - j$. This contradicts the above lower bound, so $e = f$ is actually unsatisfiable and we get the result as in the third case.
– Non-framed constraint with $e \in [\min_{base}(S)..\max_{base}(S)]$ and $f \in [\mathsf{n} - l - j - k + \max_{base}(S) + 1..\mathsf{n} - l]$, when $\star$ is $\neq$: This is the hard case, indeed we can easily obtain a set which is not $\rightrightarrows^{\mathsf{n}}$-finite. even with the constraint-irreducible extension. For instance the infinite set:

$$\begin{array}{l} 0 \neq \mathsf{n} - l \\ 0 \neq \mathsf{n} - l \wedge 0 \neq \mathsf{n} - l - 1 \\ 0 \neq \mathsf{n} - l \wedge 0 \neq \mathsf{n} - l - 1 \wedge 0 \neq \mathsf{n} - l - 2 \\ \text{etc.} \end{array}$$

is not $\rightrightarrows^{\mathsf{n}}$-finite and, contrarily to the previous cases, we cannot use the constraint-irreducible extension to simplify it. However at node α, $C_{S_{\mathcal{T}}(\alpha)}$ entails $\mathsf{n} - l - j \geq k$ (because α is aligned on $[k..\mathsf{n} - l - j]$) and thus $\mathsf{n} - l - j - k \geq 0$. On the other hand $f \geq \mathsf{n} - l - j - k + \max_{base}(S) + 1$, thus $f \geq \max_{base}(S) + 1$. So, as $e \leq \max_{base}(S)$: $f > e$. Hence the constraint $f \neq e$ is finally redundant.

Finally it is easily seen that combining all different cases preserves finiteness up to a shift by Theorem 3. Simply because by inspecting all the cases, one can see that all the expressions of a constraint inserted at a j-alignment node, are of the form $\mathsf{n} - j + q$ for some q belonging to a finite set. So all the cases are "synchronized". □

Lemma 9 (Main Lemma). *Let S be a regularly nested schema of parameter n and $\mathcal{T}$ a tableau of root schema S, then $\left\{\Pi_{S_{\mathcal{T}}(\alpha)} \middle| \alpha \text{ is an alignment node}\right\}$ is finite up to a shift on n.*

Proof. We prove that $\left\{\pi \middle| \Pi_{S_{\mathcal{T}}(\alpha)}|_p \rightsquigarrow^{\alpha}_{\mathcal{T}_p} \pi, \alpha \text{ is an alignment node}, \pi \text{ is a pattern}\right\}$ ($\mathcal{T}_p$ is the decorated of $\mathcal{T}$ w.r.t. p) is finite up to a shift on n for every position p in $\Pi_{S_{\mathcal{T}}(\alpha)}$. We get the intended result when $p = \epsilon$, indeed it is easily seen that this position is invariant by T-DPLL* hence if π is s.t. $\Pi_{S_{\mathcal{T}}(\alpha)}|_{\epsilon} \rightsquigarrow^{\alpha}_{\mathcal{T}_{\epsilon}} \pi$ then $\pi = \Pi_{S_{\mathcal{T}}(\alpha)}$ (as $\mathcal{T}_{\epsilon}$ is the decorated of $\mathcal{T}$ w.r.t. position ϵ, α may indifferently be considered as a node of $\mathcal{T}$ or a node of $\mathcal{T}_{\epsilon}$).

Let $\Pi_{S_{\mathcal{T}}(\alpha)}|_p$ be a subpattern of $\Pi_{S_{\mathcal{T}}(\alpha)}$ at some position p and π' a pattern s.t. $\Pi_{S_{\mathcal{T}}(\alpha)}|_p \rightsquigarrow^{\alpha}_{\mathcal{T}_p} \pi'$. π' is the result of applying some transformations to some other π s.t. $\Pi_{S_{\mathcal{T}}(\alpha)}|_p \rightsquigarrow^{\alpha'}_{\mathcal{T}_p} \pi$. Those transformations may be a combination of: (i) identity (if no rule applied to the subpattern between two alignment nodes), (ii) rewrite of a pattern above π, (iii) rewrite of a subpattern of π, (iv) rewrite of π itself, or (v) instantiation of a variable (in case *Unfolding* applies somewhere above π). We have to check that none of those transformations can generate an

infinite set of new schemata. This is trivial for (i). (ii) is invisible when tracing $\Pi_{S_{\mathcal{T}}(\alpha)|_p}$ (as the trace follows the moves of $\Pi_{S_{\mathcal{T}}(\alpha)|_p}$) and thus is an identity as far as we are concerned (notice that this is why tracing was designed for). For the other cases the proof goes by induction on the structure of π:

- Suppose π is a literal of index e.
  - **(iii)** Impossible.
  - **(iv)** Only *Expansion* can rewrite a literal. This is possible only if no variable other than n occurs in e, in which case *Algebraic simplification* we apply then. As seen multiple times, the introduced iterated connective will necessarily be deleted in the next alignment node (either by removing the full iteration, or by removing only the connective). Hence no schema is generated.
  - **(v)** This is possible only if there *is* a variable other than n in e (as n is never instantiated) in which case π is turned into a literal whose index does not refer to a variable other than n, then the expansion and algebraic simplifications rules apply as in Case (iv).
- Suppose $\pi = \pi_1 \vartriangle \pi_2$ where $\vartriangle \in \{\wedge, \vee\}$.
  - **(iii)** It implies that there are π_1', π_2' s.t. $\pi_1 \rightsquigarrow^{\alpha}_{\mathcal{T}_{1.p}} \pi_1'$ and $\pi_2 \rightsquigarrow^{\alpha}_{\mathcal{T}_{2.p}} \pi_2'$. By Lemmata 4 and 5 *all* expressions involving n in both π_1 and π_2 have the form $\mathsf{n}-l-j$ hence $\delta(\pi_1, \pi_2) = 0$ (where δ denotes the deviation, Section 4.1). By induction the sets of possible π_1' and π_2' are finite up to a shift, so we can apply the first corollary of Theorem 3 and conclude.
  - **(iv)** The only possible rule is *Algebraic simplification* in which case the result is obtained by induction.
  - **(v)** For every substitution σ, $\pi\sigma = \pi_1\sigma \vartriangle \pi_2\sigma$, so if $\pi \rightsquigarrow \pi\sigma$ then $\pi_1 \rightsquigarrow \pi_1\sigma$ and $\pi_2 \rightsquigarrow \pi_2\sigma$, and we conclude by induction.
- Suppose $\pi = \Delta_{\mathsf{i}=k}^{\mathsf{n}-l-j}\, \eta$ where $\Delta \in \{\bigwedge, \bigvee\}$, $j \in \mathbb{N}$. By Lemma 4, we know that every iteration must have this form.
  - **(iii)** This is handled as in the previous case except that we use the second corollary of Theorem 3 instead of the first one.
  - **(iv)** The only rewrite can be *Unfolding*. For every $p \in \mathbb{N}$, when *Unfolding* applies p times, π is turned into $\eta_1 \vartriangle \ldots \vartriangle \eta_p \vartriangle \Delta_{\mathsf{i}=k}^{\mathsf{n}-l-j-p}\, \eta$. But $\pi \rightsquigarrow^{\alpha'}_{\mathcal{T}} \eta_1$, $\ldots$, $\pi \rightsquigarrow^{\alpha'}_{\mathcal{T}} \eta_p$ so by induction hypothesis on π they all belong to the same $\rightrightarrows^{\mathsf{n}}$-finite set. So if p is big enough, there are patterns of the form η_q that will loop on each other ($q \in 1..p$). Formally there is $q_0 \in \mathbb{N}$ s.t. for every $p \in \mathbb{N}$ and every $q \in 1..p$, if $q > q_0$ then there is a $q' \leq q_0$ s.t. $\eta_q \rightrightarrows^{\mathsf{n}} \eta_{q'}$. By Lemmata 4 and 5, only iterations contain n and all of them are aligned, thus there is actually no shift on n meaning that $\eta_q = \eta_{q'}$. Hence, by *Algebraic simplification*, $\eta_1 \vartriangle \cdots \vartriangle \eta_p$ simplifies into $\eta_1 \vartriangle \cdots \vartriangle \eta_{q_0}$ at worst. Finally all schemata obtained from π are of the form $\eta_1 \vartriangle \ldots \vartriangle \eta_{q_0} \vartriangle \Delta_{\mathsf{i}=k}^{\mathsf{n}-l-j-p}\, \eta$. There are finitely many such schemata by induction hypothesis on η (and thus on $\eta_1, \ldots, \eta_{q_0}$), by the first and second corollaries of Theorem 3 (the deviation is null), and because q_0 is a constant.

(v) As $\mathsf{n} - l - j$ does not contain other variables than n it is not affected by the instantiation. All bound variables are assumed distinct so the instantiation cannot replace i. Thus, writing σ for the substitution, $\pi\sigma = \Delta_{\mathsf{i}=k}^{\mathsf{n}-l-j}(\eta\sigma)$, and we conclude by induction. □

Corollary 4. *Let S be a regularly nested schema of parameter n and $\mathcal{T}$ a tableau of root schema S, then $\{S\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha$ is an alignment node$\}$ is finite up to the constraint-irreducible and pure extensions of equality up to a shift on n.*

Proof. This follows from Definition 8 and from Theorem 3 applied to the results of Corollary 3, Lemma 8 and Main Lemma. Lemma 7 ensures that the deviation is lower than $\max_{ind}(S) - \min_{ind}(S)$. □

Theorem 4. $\mathfrak{S}$ *terminates on every regularly nested schema.*

Proof. It easily follows from the previous Corollary and the fact that $\mathfrak{S}$ uses the pure extension of equality up to a shift. Corollary 2 is also required to ensure that it is indeed sufficient to restrict ourselves to alignment nodes. □

5.3 Extensions

In the light of the previous proof, we can easily extend the class of regularly nested schemata to broader terminating classes. First we can relax a little the alignment condition:

Definition 23. *A schema S is:*

- down-aligned *iff it is framed and the frames of all iterations have the same lower bound $k \in \mathbb{Z}$ and have an upper bound of the form $\mathsf{n} - l$, where $l \in \mathbb{Z}$.*
- up-aligned *iff it is framed and the frames of all iterations have the same upper bound $\mathsf{n} - l$, where $l \in \mathbb{Z}$ and have any $k \in \mathbb{Z}$ as their lower bound.*
- broadly aligned *iff all iterations of S have frames of the form $[k_1..\mathsf{n} - k_2]$, $k_1, k_2 \in \mathbb{Z}$.*

Theorem 5. $\mathfrak{S}$ *terminates on every schema which is monadic, of limited progression and down-aligned.*

Proof. (Sketch) Such a schema is almost regularly nested except that down-alignment is substituted to alignment. It is easily seen that, after the first passing in step 2, either the constraint $k \leq \mathsf{n} - l$ or $k > \mathsf{n} - l$ has been added to the node, where $l = \min\{l' \mid \mathsf{n} - l'$ is the upper bound of an iteration in $S\}$. If it is $k \leq \mathsf{n} - l$ then it implies that $k \leq \mathsf{n} - l'$ for every $l' \geq l$. In step 3, all iterations are unfolded until no longer possible. Hence here, all iterations will be unfolded until their upper bound reaches $\mathsf{n} - l - 1$ (even those of frames $[k..\mathsf{n} - l']$, $l' > l$). As a consequence all iterations are now aligned and we are back in the same case as for regularly nested schemata. We call this phase, where all iterations progressively become aligned, the *rectification*. Rectification terminates because of a similar argument to the one proving the termination of Step 3 in the proof

of Lemma 6. In the case where $k > \mathsf{n} - l$ has been added, it is easily seen that there will be finitely many unfoldings of iterations of frame $[k..\mathsf{n} - l']$, $l' > l$ (actually there will be at most $m - l'$ such unfoldings per iteration, where $m = \max\{l' \mid \mathsf{n} - l'$ is the upper bound of an iteration in $S\}$) then all iterations will be empty. □

Theorem 6. $\mathfrak{S}$ *terminates on every schema which is monadic, of limited progression and up-aligned.*

Proof. (Sketch) In this case schemata will, in general, never become aligned: suppose we have two iterations $\Delta_{\mathsf{i}=k_1}^{\mathsf{n}-l}\,\pi$ and $\nabla_{\mathsf{j}=k_2}^{\mathsf{n}-l}\,\pi'$ with $k_1 < k_2$. Then any constraint $k_1 \geq \mathsf{n} - l - j$ implies $k_2 \geq \mathsf{n} - l - j - k_1 + k_2$ so when $\Delta_{\mathsf{i}=k_1}^{\mathsf{n}-l}\,\pi$ will be unfolded until $\mathsf{n} - l - j$, $\nabla_{\mathsf{j}=k_2}^{\mathsf{n}-l}\,\pi'$ will be unfolded until $\mathsf{n} - l - j - k_1 + k_2$. We will never reach alignment. However it is easily seen that the difference between two upper bounds (here $k_2 - k_1$) will always remain lower than the deviation of the original schema. Hence slight modifications in the proof of Main Lemma enable to conclude. The hard point lies in the application of *Algebraic simplification* in the item (iv) of the iteration case, indeed now we cannot conclude from $\pi'_{f-k+q} \rightrightarrows^{\mathsf{n}} \pi'_{f-k+q'}$ that $\pi'_{f-k+q} = \pi'_{f-k+q'}$ as there is no alignment. However as the "misalignment" is confined to a finite set, the sequence $(\pi'_{f-k+1} \vartriangle \cdots \vartriangle \pi'_f)_{k\in\mathbb{N}}$ still cannot grow infinitely. □

Theorem 7. $\mathfrak{S}$ *terminates on every schema which is monadic, of limited progression and broadly-aligned.*

Proof. (Sketch) This proof is close to the previous one. Actually we do not really need the fact that the upper bound is the same in the previous proof. □

Definition 24. *A schema S is:*

- variable-aligned *on* $[e_1..e_2]$, *for two linear expressions* e_1, e_2 *iff every iteration of S is framed either on* $[e_1..e_2]$, *or on* $[e_1..\mathsf{i} + q]$ *where* i *is a non-parameter variable and* $q \in \mathbb{Z}$.
- simply variable-aligned *iff it is variable-aligned and* $q = 0$.
- positively variable-aligned *iff it is variable-aligned and* $q \geq 0$.
- negatively variable-aligned *iff it is variable-aligned and* $q \leq 0$.
- broadly variable-aligned *iff all iterations of S have frames of the form* $[k_1..\mathsf{n} - k_2]$, *or* $[k_1..\mathsf{i} - k_2]$, *where* $k_1, k_2 \in \mathbb{Z}$.

An iteration of frame $[e_1..\mathsf{i} + q]$ *is called an* i-iteration. *Let* $\mathfrak{S}'$ *be the strategy* $\mathfrak{S}$ *except that* Emptiness *is disallowed.*

Theorem 8. $\mathfrak{S}'$ *terminates on every schema which is monadic, of limited progression and simply variable-aligned on* $[k..\mathsf{n} - l]$ *for some* $k, l \in \mathbb{Z}$.

Proof. (Sketch) It is easily seen that variable-alignment is preserved all along the procedure (this fact plays the same role as Lemma 4): indeed, the only way an i-iteration $\Delta_{\mathsf{j}=k}^{\mathsf{i}}\,\pi$ may be unfolded is by unfolding the iteration binding i (which

necessarily exists as i is not a parameter). Let us write it $\nabla^{e}_{\mathsf{i}=k}\,\pi'$, e is either a non-parameter variable or a linear expression of the form $\mathsf{n}-l-j$. When this iteration is unfolded, it is turned into $\nabla^{e-1}_{\mathsf{i}=k}\,\pi' \triangledown \pi[e/\mathsf{i}]$. We have now two copies of $\Delta^{\mathsf{i}}_{\mathsf{j}=k}\,\pi$: one inside $\nabla^{e-1}_{\mathsf{i}=k}\,\pi'$, and one inside $\pi[e/i]$. The last one has actually been instantiated: $\Delta^{e}_{\mathsf{j}=k}\,\pi$. As this iteration has the same frame as $\nabla^{e}_{\mathsf{i}=k}\,\pi'$ it also meets the requirements to be unfolded, which indeed happens, turning the iteration into $\Delta^{e-1}_{\mathsf{j}=k}\,\pi$. This new iteration is framed on $[k..e-1]$ like every other non i-iteration in the node. As *Emptiness* is disallowed all non-instantiated i-iterations are kept as is. Finally there is a finite number of such instantiations as each time the number of iterations below the observed iteration decreases. As a consequence all generated schemata are translated w.r.t. n, and the proof is then very similar to the regularly nested case. □

Theorem 9. *$\mathfrak{S}'$ terminates on every schema which is monadic, of limited progression and positively variable-aligned on $[k..\mathsf{n}-l]$ for some $k, l \in \mathbb{Z}$.*

Proof. (Sketch) It is a combination of the previous proof and proof of Theorem 5. Except that now rectification not only occurs at the beginning of the procedure but each time an i-iteration is unfolded. Indeed each time i is instantiated in an i-iteration $\Delta^{\mathsf{i}+q}_{\mathsf{j}=k}\,\pi$, this iteration has to be rectified. There are still finitely many schemata that are generated as instantiating i-iterations can only lead to finitely many different iterations up to a shift. □

Theorem 10. *$\mathfrak{S}'$ terminates on every schema which is monadic, of limited progression and negatively variable-aligned on $[k..\mathsf{n}-l]$ for some $k, l \in \mathbb{Z}$.*

Proof. (Sketch) It is a combination of the proofs of Theorems 8 and 6. Except that now the maximum deviation used in Theorem 3 will not be the deviation of the original schema S, but rather the deviation of S in which *all iterations have been unfolded once*. Indeed schemata are not aligned anymore, even after rectification: when Step 2 terminates, the constraint $k < \mathsf{n}-l-\mathsf{j}$ where $l = \min\{l' \mid \mathsf{n}-l' \text{ is the upper bound of an iteration in } S\}$ has been added. Hence if an i-iteration $\Delta^{\mathsf{i}-q}_{\mathsf{j}=k}\,\pi$, $q>0$, is instantiated, we get $\Delta^{\mathsf{n}-l-j-q}_{\mathsf{j}=k}\,\pi$ which cannot be unfolded as nothing ensures that $k \leq \mathsf{n}-l-j-q$. So we have to deal with mis-alignment. As in the proof of Theorem 6, it is easily seen that this is not a problem as we have a maximum deviation as noted above. □

Finally the following theorem is obtained by combining all previous proofs:

Theorem 11. *$\mathfrak{S}'$ terminates on every schema which is monadic, of limited progression and broadly variable-aligned.*

6 Conclusion

We have presented a proof procedure, called DPLL*, for reasoning with propositional formula schemata. The main originality of our calculus is that the inference rules may apply at a deep position in a formula, a feature that is essential for

handling nested iterations. A looping mechanism is introduced to improve the termination behavior. We defined an abstract notion of looping which is very general, then instantiated this relation into a more concrete version that is decidable, but still powerful enough to ensure termination in many cases.

We identified a class of schemata, called regularly nested schemata, for which DPLL* always terminates. This class is much more expressive than the class of regular schemata handled in [2]. The principle of the termination proof is (to the best of our knowledge) original: it goes by investigating how a given subformula is affected by the application of expansion rules on the "global" schema. This is done by defining a "traced" version of the calculus in which additional information is provided concerning the evolution of a specific subformula (or set of subformulae, since a formula may be duplicated). This also required a thorough investigation of the properties of the looping relation. We believe that these ideas could be reused to prove termination of other calculi, sharing common features with DPLL* (namely calculi that operate at deep levels inside a formula and that allow cyclic proofs).

We do not know of any similar work in automated deduction. Schemata have been studied in logic (see e.g. [11, 3, 17]) but our approach is different from these (essentially proof theoretical) works both in the particular kind of targeted schemata and in the emphasis on the automation of the proposed calculi. However one can find similarities with other works.

Iterations can obviously recall of fixed-point constructions, in particular in the (modal) μ-calculus[5] [5] (with $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \phi$ translated into something like $\mu X.\phi \wedge X$). However the semantics are very different: that of iterated schemata is restricted to *finite models* (since every parameter is mapped to an integer, the obtained interpretation is finite), whereas models of the μ-calculus may be infinite. Hence the involved logic is very different from ours and actually simpler from a theoretical point of view: the μ-calculus admits complete proof procedures and is decidable, whereas schemata enjoy none of those properties. The relation between schemata and the μ-calculus might actually be analogous to the relation between finite model theory [13] and classical first-order logic. The detailed comparison of all those formalisms is worth investigating but out of the scope of the present work. Other fixed-point logics exist that can embed schemata such as least fixpoint logic [16] or the first-order μ-calculus [18]. However they are essentially studied for their theoretical properties i.e. complete or decidable classes are seldom investigated. Actually the only such study that we know of is in [4] and iterated schemata definitely do not lie in the studied class nor can be reduced to it.

One can also translate schemata into first-order logic by turning the iterations into (bounded) quantifications i.e. $\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \phi$ (resp. $\bigvee_{\mathsf{i}=1}^{\mathsf{n}} \phi$) becomes $\forall \mathsf{i}(1 \leq \mathsf{i} \leq \mathsf{n} \Rightarrow \phi)$ (resp. $\exists \mathsf{i}(1 \leq \mathsf{i} \leq \mathsf{n} \wedge \phi)$). This translation is completed by quantifying universally on the parameters and by axiomatizing first-order linear arithmetic. Then automated reasoning is achieved through a first-order theorem prover. As arithmetic is involved, useful results would probably be obtained only with in-

[5] In which many temporal logics e.g. CTL, LTL, and CTL* can be translated.

ductive theorem provers [9, 7]. However there are very few decidability results that can be used with such provers. Moreover most of those systems are designed to prove formulae of the form $\forall \boldsymbol{x}.\phi$ where ϕ is *quantifier-free*. The translation sketched above clearly shows most translated schemata do not match this form. Actually this is already the case of any schema involving only one iterated *disjunction*. Indeed adding existential quantification in inductive theorem proving is known to be a difficult problem. Notice finally that this translation completely hides the *structure* of the original problem.

Finally, as we have seen in Section 4, decidability of regularly nested schemata lies in the detection of *cycles* during the proof search. This idea is not new, it is used e.g. in tableaux methods dealing with modal logics in transitive frames [14], or μ-calculi [8]. However our cycle detection is quite different because we use it to actually prove by induction. Notice in particular that, contrarily to the mentioned tableaux methods, we cannot in general ensure termination. It is more relevant to consider our case as a particular instance of *cyclic proofs*, which are studied in proof theory precisely in the context of proofs by induction. Both [6] and [20] show that cyclic proofs seem as powerful as systems dealing classically with induction. A particular advantage of cyclic proofs is that finding an invariant is not needed, making them particularly suited to automation. This is also extremely useful for the formalization of mathematical proofs, because it allows one to express a potentially infinite proof steps sequence, thus avoiding the explicit use of the induction principle. This last feature has been used to avoid working with more expressive logical formalisms [15]. However once again studies on cyclic proofs are essentially theoretical and no complete class is identified at all.

Future work includes the implementation of the DPLL* calculus and the investigation of its practical performances[6]. It would also be interesting to extend the termination result in Section 5 to non monadic schemata so as to be able to express e.g. the binary multiplier of the Introduction. Extension of the previous results to more powerful logics (such as first-order logic or modal logic) naturally deserves to be considered. Finally the proof of Theorem 4 seems to be a powerful tool. We hope that the underlying ideas could be useful in other proof systems. In particular investigating more thoroughly the looping relation could give rise to interesting connections.

References

1. V. Aravantinos, R. Caferra, and N. Peltier. A DPLL proof procedure for propositional iterated schemata. In *Workshop Structures and Deduction, Proceedings of the European Summer School in Logic, Language and Information*, 2009.
2. V. Aravantinos, R. Caferra, and N. Peltier. A Schemata Calculus For Propositional Logic. In *18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2009)*, LNCS. Springer, 2009.

[6] An implementation of the less powerful but simpler STAB procedure is available at `http://regstab.forge.ocamlcore.org`.

3. M. Baaz and R. Zach. Short proofs of tautologies using the schema of equivalence. In *Computer Science Logic (CSL '93)*. Springer-Verlag, 1994. LNCS 832.
4. David Baelde. On the proof theory of regular fixed points. In *Proceedings of the* 18^{th} *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2009)*, LNCS. Springer, 2009.
5. J. Bradfield and C. Stirling. Modal Mu-Calculi. In P. Blackburn, J. F. A. K. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2007.
6. James Brotherston. Cyclic Proofs for First-Order Logic with Inductive Definitions. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of TABLEAUX 2005*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.
7. Alan Bundy. The Automation of Proof by Mathematical Induction. In Robinson and Voronkov [19], pages 845–911.
8. R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Inf.*, 27(9):725–747, 1990.
9. H. Comon. Inductionless induction. In Robinson and Voronkov [19], chapter 14.
10. D.C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence 7*. Edinburgh University Press, 1972.
11. John Corcoran. Schemata: the concept of schema in the history of logic. *The Bulletin of Symbolic Logic*, 12(2):219–240, June 2006.
12. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Communication of the ACM*, 5:394–397, 1962.
13. Ronald Fagin. Finite-Model Theory - A Personal Perspective. *Theoretical Computer Science*, 116:3–31, 1993.
14. Rajeev Goré. Chapter 6: Tableau Methods for Modal and Temporal Logics. In M D'Agostino, D Gabbay, R Hähnle, J Posegga, editor, *Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers, 1999. `http://arp.anu.edu.au/~rpg` (draft).
15. Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. Proof analysis with HLK, CERES and ProofTool: Current status and future directions. In Schulz S. Sutcliffe G., Colton S., editor, *Workshop on Empirically Successful Automated Reasoning for Mathematics (ESARM)*, pages 21–41, July 2008.
16. Neil Immerman. Relational queries computable in polynomial time (Extended Abstract). In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 147–152, New York, NY, USA, 1982. ACM.
17. V. P. Orevkov. Proof schemata in Hilbert-type axiomatic theories. *Journal of Mathematical Sciences*, 55(2):1610–1620, 1991.
18. David Michael Ritchie Park. Finiteness is Mu-ineffable. *Theoretical Computer Science*, 3:173–181, 1976.
19. John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
20. C. Sprenger and M. Dam. On the Structure of Inductive Reasoning: Circular and Tree-shaped Proofs in the mu-Calculus. In *Proc. FOSSACS'03*, Springer LNCS, pages 425–440, 2003.

A An Example of a DPLL* Proof

We want to prove that $A+0=A$ where $+$ denotes the addition specified by the schema *Adder* described in the Introduction. A SAT-solver can easily prove this for a fixed n (say $\mathsf{n}=9$). We show how to prove it for all $\mathsf{n}\in\mathbb{N}$ with DPLL*. This simple example has been chosen for the sake of conciseness, but commutativity or associativity of the adder could have been proven too.

We express the fact that the second operand is null:

$$\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \neg B_{\mathsf{i}}$$

and the conjecture i.e. the fact that the result equals the first operand:

$$\bigwedge_{\mathsf{i}=1}^{\mathsf{n}} A_{\mathsf{i}} \Leftrightarrow S_{\mathsf{i}}$$

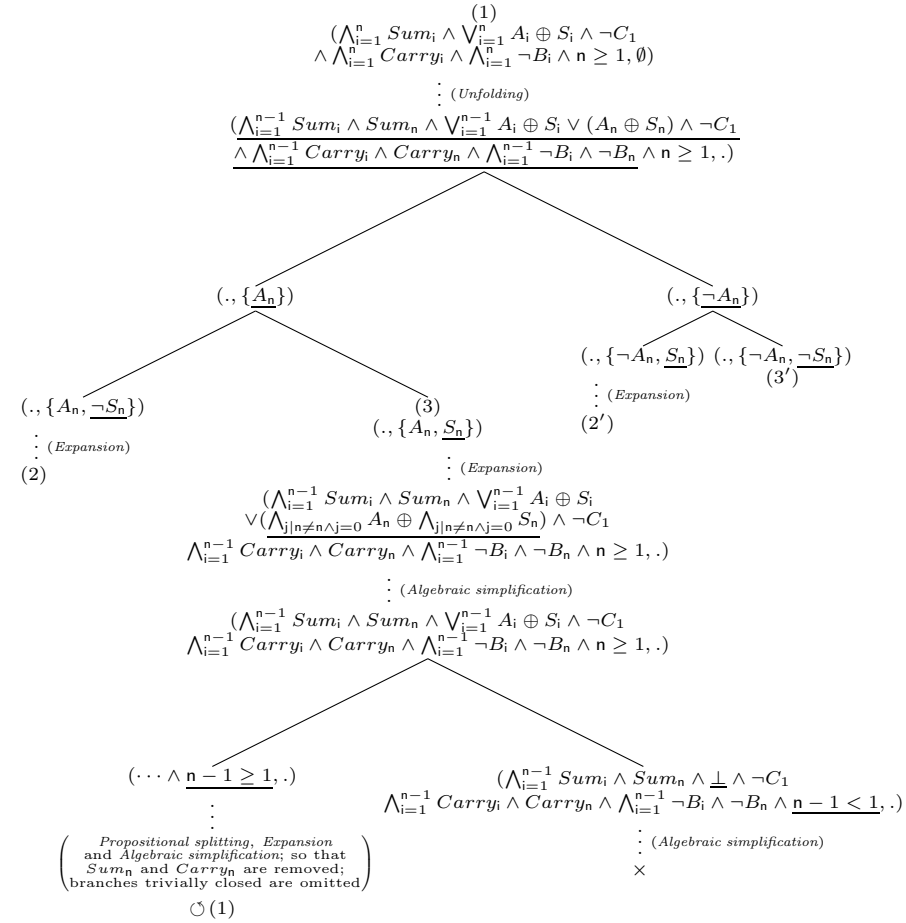
We negate the conjecture in order to prove it by refutation:

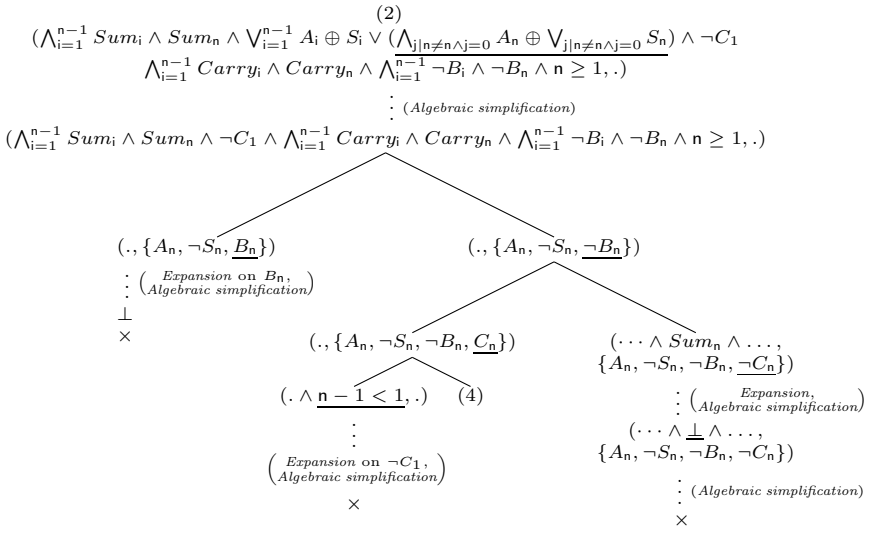
$$\bigvee_{\mathsf{i}=1}^{\mathsf{n}} A_{\mathsf{i}} \oplus S_{\mathsf{i}}$$

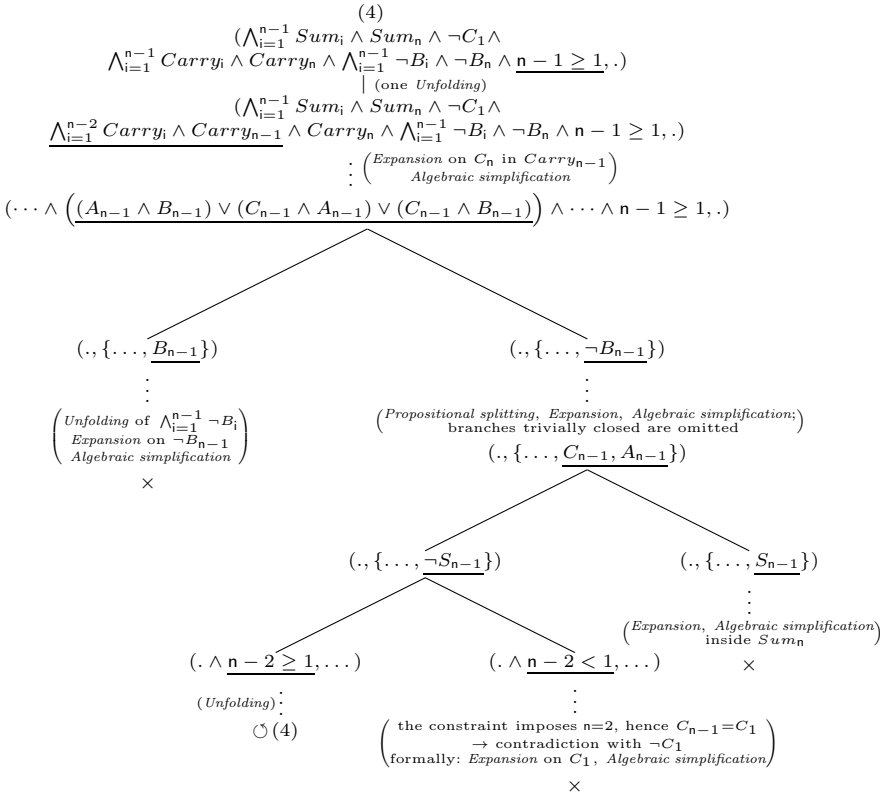
Finally we want to refute:

$$\mathit{Adder} \wedge \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \neg B_{\mathsf{i}} \wedge \bigvee_{\mathsf{i}=1}^{\mathsf{n}} A_{\mathsf{i}} \oplus S_{\mathsf{i}}$$

The following figure is only a sketch of the real tableau: several rules are often applied at once, denoted by vertical dots labelled with the names of the used rules. We use the conventions that closed leaves are marked by $\times$, leaves looping on a node α by $\circlearrowleft(\alpha)$. Changed parts of a node are underlined, "." means "same value as parent node's". We recall that $S_1 \Leftrightarrow S_2$ and $S_1 \oplus S_2$ are shorthands for $(S_1 \Rightarrow S_2) \wedge (S_2 \Rightarrow S_1)$ and $\neg(S_1 \Leftrightarrow S_2)$ respectively. All bound variables should be renamed so as to have different names, this is not done for the sake of readability.


(1)
$(\bigwedge_{i=1}^{n} Sum_i \wedge \bigvee_{i=1}^{n} A_i \oplus S_i \wedge \neg C_1$
$\wedge \bigwedge_{i=1}^{n} Carry_i \wedge \bigwedge_{i=1}^{n} \neg B_i \wedge n \geq 1, \emptyset)$
(Unfolding)
$(\underline{\bigwedge_{i=1}^{n-1} Sum_i \wedge Sum_n \wedge \bigvee_{i=1}^{n-1} A_i \oplus S_i \vee (A_n \oplus S_n)} \wedge \neg C_1$
$\underline{\wedge \bigwedge_{i=1}^{n-1} Carry_i \wedge Carry_n \wedge \bigwedge_{i=1}^{n-1} \neg B_i \wedge \neg B_n \wedge n \geq 1}, .)$
$(., \{\underline{A_n}\})$
$(., \{\underline{\neg A_n}\})$
$(., \{\neg A_n, \underline{S_n}\})$
$(., \{\neg A_n, \underline{\neg S_n}\})$
(3′)
(Expansion)
(2′)
$(., \{A_n, \underline{\neg S_n}\})$
(Expansion)
(2)
(3)
$(., \{A_n, \underline{S_n}\})$
(Expansion)
$(\bigwedge_{i=1}^{n-1} Sum_i \wedge Sum_n \wedge \bigvee_{i=1}^{n-1} A_i \oplus S_i$
$\vee (\underline{\bigwedge_{j|n\neq n\wedge j=0} A_n \oplus \bigwedge_{j|n\neq n\wedge j=0} S_n}) \wedge \neg C_1$
$\bigwedge_{i=1}^{n-1} Carry_i \wedge Carry_n \wedge \bigwedge_{i=1}^{n-1} \neg B_i \wedge \neg B_n \wedge n \geq 1, .)$
(Algebraic simplification)
$(\bigwedge_{i=1}^{n-1} Sum_i \wedge Sum_n \wedge \bigvee_{i=1}^{n-1} A_i \oplus S_i \wedge \neg C_1$
$\bigwedge_{i=1}^{n-1} Carry_i \wedge Carry_n \wedge \bigwedge_{i=1}^{n-1} \neg B_i \wedge \neg B_n \wedge n \geq 1, .)$
$(\cdots \wedge \underline{n - 1 \geq 1}, .)$
(Propositional splitting, Expansion and Algebraic simplification; so that Sum_n and $Carry_n$ are removed; branches trivially closed are omitted)
↺ (1)
$(\bigwedge_{i=1}^{n-1} Sum_i \wedge Sum_n \wedge \underline{\bot} \wedge \neg C_1$
$\bigwedge_{i=1}^{n-1} Carry_i \wedge Carry_n \wedge \bigwedge_{i=1}^{n-1} \neg B_i \wedge \neg B_n \wedge \underline{n - 1 < 1}, .)$
(Algebraic simplification)
×


(2)
$(\bigwedge_{i=1}^{n-1} Sum_i \wedge Sum_n \wedge \bigvee_{i=1}^{n-1} A_i \oplus S_i \vee (\underline{\bigwedge_{j|n\neq n\wedge j=0} A_n \oplus \bigvee_{j|n\neq n\wedge j=0} S_n}) \wedge \neg C_1$
$\bigwedge_{i=1}^{n-1} Carry_i \wedge Carry_n \wedge \bigwedge_{i=1}^{n-1} \neg B_i \wedge \neg B_n \wedge n \geq 1, .)$
(Algebraic simplification)
$(\bigwedge_{i=1}^{n-1} Sum_i \wedge Sum_n \wedge \neg C_1 \wedge \bigwedge_{i=1}^{n-1} Carry_i \wedge Carry_n \wedge \bigwedge_{i=1}^{n-1} \neg B_i \wedge \neg B_n \wedge n \geq 1, .)$
$(., \{A_n, \neg S_n, \underline{B_n}\})$
(Expansion on B_n, Algebraic simplification)
⊥
×
$(., \{A_n, \neg S_n, \underline{\neg B_n}\})$
$(., \{A_n, \neg S_n, \neg B_n, \underline{C_n}\})$
$(\cdots \wedge Sum_n \wedge \ldots,$
$\{A_n, \neg S_n, \neg B_n, \underline{\neg C_n}\})$
$(. \wedge \underline{n - 1 < 1}, .)$
(4)
(Expansion, Algebraic simplification)
$(\cdots \wedge \underline{\bot} \wedge \ldots,$
$\{A_n, \neg S_n, \neg B_n, \neg C_n\})$
(Expansion on $\neg C_1$, Algebraic simplification)
(Algebraic simplification)
×
×



(2′) and (4′) are very similar to (2) and (4).